**Juris Borzovs**
**Dr.sc.comp.**

# HABILITĀCIJAS DARBU KOPUMS

## Informācijas tehnoloģijas disciplinēšana Latvijā

**Rīga, 1999**

# BASIC REQUIREMENTS FOR NATIONAL INFORMATION SYSTEMS

**Juris Borzovs**
*SWH Institute of Technology*
**Gints Ernestsons**
*Citizenship and Immigration Department, Ministry of Interior Affairs*
**Ints Lukss**
*Land Cadastre of Latvia*
**Imants Mētra**
*Information Centre of Civil Registration System*

Almost 10 national information systems are at different stages of maturity today in Latvia: population register, enterprises register, land and equities registers, etc. Due to several reasons, such as very tight national budget, different approaches enforced by foreign technical aid, lack of national co-ordinating or governmental body in information technology (IT), even taste of particular developer, etc., Latvia has no officially adopted national program in IT. This makes a huge barrier in integration of national and, evidently, of regional, Baltic or Nordic, IT-systems. In such a disintegrated and uncoordinated environment the starting point for ordering of chaos could be an approved set of basic requirements over software and hardware. Draft proposal for the set formed of different levels of abstraction is presented in this paper.

## 1. Future of Information System Facilities

To characterise information systems (IS) facilities in the near future let us consider the following features[1]:

- Physically separated elements of the facilities will be interconnected over public networks.
- In team environments, there will be a transition from physical to logical integration.
- There will be integration of multimedia.
- In facilities networking will be fundamental to facility design, i.e. fax, e-mail, etc.
- Facilities will ease the invoking support services, e.g. copying.
- Facilities will be designed considering the ergonomic arrangement of computer displays, input devices, processors, storage units, and output devices.
- Facilities will contain disaster recovery features such as emergency power for graceful degradation of computer systems and automatic switch over of the networks to secondary networks.
- Facilities will satisfy the security objectives.
- Facilities will be equipped to receive products using electronic data transfer instead of portable media transfers.

## 2. The Approach and Level-1 Requirements

We assume top-down and role-oriented approach to be useful during analysis and presentation of requirements over emerging national information systems. This means the following:

- requirements are developed beginning with very obvious informal ones, e.g. *nobody wants to learn;*
- requirements are stated from different points-of-view of persons engaged in IT, e.g. end-users, government officials, etc.;
- requirements are developed using different levels of abstraction, i.e. those informal requirements must be understandable for everybody and make Level-1 requirements, more detailed Level-2 requirements are derived from Level-1 requirements, etc. Level-3 or may be Level-4 requirements represent precise statements, e.g. standards, directives; the lowest level requirements must be available in full text, not only through reference.

Such an approach should be useful not only for disciplined analysis of problem but also and especially in presentation to decision-makers of different level of expertise. The requirements when adopted will represent a framework for IT planning and development of national IS.

It goes without saying that IS of different criticality and area of application must conform to the requirements differently. In fact, there is a need to formulate precise meaning of the requirement, possible metric and its corresponding appropriate values, etc. Below one can find more precise requirements regarding government and other national IS. The other state-owned institutions' IS should follow them at the extent of needed interoperability with government systems.

Let start from level-1 requirements set:

1. End-user's requirements
   - 1.1 Ease of learn
   - 1.2 The same and simple mode of dialogue
   - 1.3 Information exactly in time, at any location, in exactly needed details
   - 1.4 Integral use of different IS
   - 1.5 Data privacy and security
2. IS-customer's requirements
   - 2.1 Minimal cost/benefit level
   - 2.2 Step-wise development of IS
   - 2.3 Easy integration of newly installed software or hardware with existing one
   - 2.4 Licensed software
   - 2.5 Adherence to standards
3. Government specific requirements
   - 3.1 Appropriate government security and audit level
   - 3.2 Ability of IS regional Baltic, Nordic, etc. integratio..
4. Software incl. maintenance manager's requirements
   - 4.1 Reuse of existing software
   - 4.2 Portability to different hardware platforms
   - 4.3 Maintainability
   - 4.4 Ease of co-ordination between different developers, current as well as former and subsequent
   - 4.5 Ability of unbiased testing and test coverage evaluation
   - 4.6. Readability and understandability of project documentation

- 4.7 Adherence to standards

5. Hardware manager's requirements
- 5.1 Use of quality, reliable, easy-to-repair hardware
- 5.2 Use of interchangeable, interoperable, widely-used hardware
- 5.3 Adherence to standards

6. Data administrator's requirements
- 6.1 Data privacy
- 6.2 Data security
- 6.3 Data completeness
- 6.4 Data transferability
- 6.5 Data updatability
- 6.6 Data correctness

## 3. Requirements over National Information Systems (NIS)

The authors propose a 3-level structure of requirements. The first, top level, requirements should be expressed in plain terms to be understandable by each official, professional and user, e.g. *user has not to learn much*. The second level requirements describe possible ways to achieve the top level needs, e.g. *user interface must conform to GUI conventions*, and establish goals for the third level requirements. The latter should be precise, related to standards or approved conventions, and mainly are not understandable by non-professionals.

It should be nice to obtain a tree of requirements where every higher level requirement is expressed by one or more lover level requirements without duplication. As soon as different views of parties involved in NIS likely interleave and sometimes contradict construction of the tree is not straightforward if even possible. Therefore we present below only semistructured system of requirements over NIS. L1, L2, L3 stand for level-1, level-2, level-3 requirements, respectively. The requirements are grouped in 6 groups representing views of different parties involved in NIS.

1. End-user's requirements

| | |
|---|---|
| • 1.1 Ease of learn | L1 |
| • 1.2 The same and simple mode of dialogue | L1 |
| - stability | L2 |
| - visuality of functions | L2 |
| - based on pictures | L2 |
| - standard or common usage | L2 |
| - easily available manuals | L2 |
| - freedom of choice | L2 |
| - CUA - common user access | L3 |
| - GUI type standards - Window/Motif/Xwindows | L3 |
| - icon based | L3 |
| - GUI based OS | L3 |
| • 1.3 Information exactly in time, at any location, in exactly needed details | L1 |
| - information in acceptable time | L2 |
| - terminals on tables of users | L2 |
| - possibility to work from geographically different places | L2 |
| - systems must allow to formulate correct requests for data retrieval and get exact answers | L2 |

- mobile computing L2
  - on line systems everywhere L3
  - networking at any location L3
  - wide internetworking L3
  - SQL-client-server systems L3
- 1.4 Integral use of different IS L1
  - possibility to switch between databases from the same terminal L2
  - common user access interface to all applications L2
  - possibility to work with multiple applications simultaneously L2
  - no data losses in transactions between systems L2
    - client-server systems (ORACLE, Informix, Sybase, etc.) L3
    - SQL data retrieval standard L3
    - multi-user OS at server sites - UNIX, Windows NT L3
    - multitasking OS at user workplace - Windows, OS/2, MAC OS L3
    - possibility to LOGIN on multiple remote sites at once L3
    - distributed RDBMS L3
    - two-phase commit transaction mechanism L3
- 1.5 Data privacy and security L1
  - protection of private data of users L2
  - assurance of user data safety L2
  - no other users should fool the system L2
  - privacy of correspondence L2
  - monitoring who uses data L2
  - no changes or forging of data without sanction L2
  - easy security administration - adding new users, changing their rights, regular audit reports L2
    - user access restrictions according their rights L3
    - access rights to databases, applications, administration L3
    - secure password system L3
    - C2-level conforming security standards L3
    - DSA level or higher private data encoding L3
    - public key cryptography in mail exchange L3

2. IS-customer's requirements

- 2.1 Minimal cost/benefit level L1
  - contests for system development projects L2
  - use of standards at every level L2
  - exclusion of parallelism L2
    - co-ordinating body or authority L3
- 2.2 Step-wise development of IS L1
  - gradual development of systems L2
  - new implementations inherit previous efforts L2
  - revision on economically justified basis L2
  - expandability of previous software/hardware L2
  - independence of project documentation from software/hardware platforms L2
    - CASE-development L3
    - use of portable software L3
    - adherence to standards L3
- 2.3 Easy integration of newly installed software or hardware with existing one L1
  - no special knowledge needed to install new systems L2

- support for previous data models and databases L2
- portability of software L2
- interoperability of hardware on standards basis L2
- flexible and expandable hardware solutions L2

• 2.4 Licensed software L1
- purchase everything legally L2
- receive appropriate warranties L2
- receive punishment for misbehaviour L2
  - surveillance authority needed L3
  - legal procedures and regulations needed L3
  - established court-based proceedings L3

• 2.5 Adherence to standards L1
- use national standards everywhere L2
- if there is no national, use international standards L2
- if there are no standards use widely accepted solutions L2

3. Government specific requirements

• 3.1 Appropriate government security and audit level L1
- legal basis for security requirements L2
- responsibility for data misuse L2
- possibility to monitor private data users L2
- responsibility of officials L2
  - legislation on public registers and private data protection L3
  - executive supervision body established L3
  - regulations and obligations for every authority L3

• 3.2 Ability of IS regional, Baltic, Nordic, etc. integration L1
- co-ordination of The Requirements with other regional, Baltic, Nordic, etc. countries L2
- regular exchange of information L2
  - state-authorised organisation representing Latvia internationally in everything regarding IT L3
  - official acceptance of international standards L3
  - participation in international standardisation bodies L3

4. Software incl. maintenance manager's requirements

• 4.1 Reuse of existing software L1
- no repeated programming necessary for already done job L2
- easy reuse of efforts made by previous programmers L2
- use software providing backup compatibility L2
  - object-oriented 4GL-development tools L3
  - inheritance of software L3
  - CASE-based system design L3

• 4.2 Portability to different hardware platforms L1
- change of equipment producer should not lead to new purchase of the same software L2
- end-user systems should run on different hardware platforms L2
- use portable environment L2
  - portable operating systems: UNIX, Windows NT L3

• 4.3 Maintainability L1
- easy installation L2
- easy version control L2
- easy documentation L2

- automatic software-based installation L3
- environment self-detection and versioning L3
- automatic resolving of conflicts between different systems L3
- on-line help L3
- CD-ROM based hypertext documentation L3
- complete uninstall feature L3

- 4.4 Ease of co-ordination between different developers, current as well as former and subsequent L1
  - efficient teamwork for a development project L2
  - no project information losses when person move away L2
  - consistency of documentation along all project life L2
  - easily accessible project information L2
    - CASE-based development L3
    - priority of graphic models and schemes over programming techniques L3
    - automatic versioning control L3
    - SQL-based repository of project information L3
- 4.5 Ability of unbiased testing and test coverage evaluation L1
  - production system should be without errors L2
  - all errors should be reported immediately L2
  - new and corrected modifications of the system should be implemented without interruption of production L2
    - automatic test tools L3
    - standard test methodology L3
    - develop and test prototypes first L3
    - use third-party testing L3
- 4.6. Readability and understandability of project documentation L1
  - all project documentation should be easily accessible and understandable L2
  - ensure consistency in documentation L2
  - reflect current status of the system correctly L2
  - more pictures, less text L2
    - CASE-development L3
    - use DEMO and training examples L3
    - use graphics of models instead of algorithms L3
- 4.7 Adherence to standards L1
  - authority should ensure that projects are accepted only in case of strong standards adherence L2

5. Hardware manager's requirements

- 5.1 Use of quality, reliable, easy-to-repair hardware L1
  - demand warranty more than one year L2
  - use hardware containing less components with more functionality L2
  - use hardware requiring no special engineering knowledge L2
  - use modular, easy to expand hardware L2
  - use hardware conforming to standards L2
    - on-chip high packaging level technology L3
    - modular snap-together components L3
    - no jumpers or switches needed to configure L3
    - upgradability in modular additions L3

- 5.2 Use of interchangeable, interoperable, widely-used hardware L1
  - use hardware suppliers producing standard equipment and components L2
  - don't use propriety equipment L2
  - don't stick to one manufacturer L2
    - use bus standards like ISA/EISA/PCI in personal computers, VMEbus in mid-range computers L3
    - use device standards like SCSI-2 L3
    - use communication devices supporting X.25, TCP/IP L3
- 5.3 Adherence to standards
  - see 2.5 L2
    - see 2.5 L3

6. Data administrator's requirements (requirements not yet refined at lower levels)

- 6.1 Data privacy L1
- 6.2 Data security L1
- 6.3 Data completeness L1
- 6.4 Data transferability L1
- 6.5 Data updatability L1
- 6.6 Data correctness L1

It is necessary to make revisions into IS requirements regularly, say, once a year, especially the lowest level ones. Revisions should reflect technology advances and ensure acceptance of progressive conceptual models in government IS.

## References

[1] Master Plan for Software Engineering Standards, Version 0.8, IEEE Software Engineering Standards Long-Range Planning Group, March 2,1993.

### Appendix. Excerpts from "Directives for National Automated Information Systems"

To the moment of submission of this paper a proposal for "Directives for National Automated Information Systems" is submitted to the Prime-Minister. We present below an excerpts from the Directives (unofficial translation). The Directives is a concise version of the Requirements considered above.

---

To fulfil needs of IS users, implementation of national information system must adhere to the following technical requirements:

1. Architecturally, a system must be technically "open". It means that every hardware device must be interchangeable with components of other vendors when they conform to widely-used standards.
   - 1.1. ISA/EISA/VESA/PCI/VMEbus computers.
   - 1.2. SCSI, Fast-SCSI I/O device interfaces.
   - 1.3. RS-232 serial and Centronics parallel interfaces.
2. Functionally, a system must be multitasking and multi-user one. It must be run on powerful central mainframe under appropriate operating system.
   - 2.1. RISC multiprocessor base with UNIX System V Release 4 operating system.
3. Workstations are represented by personal computers, possibly included in LAN.
   - 3.1. IBM PCs under DOS/Windows/Novell operating systems or local UNIX operating system. LAN standard is Ethernet.
4. Information systems must have client-server architecture. To retrieve information, SQL ANSI standard must be used.
   - 4.1. ORACLE7, Sybase, Informix-Online.
5. Systems must be integrated via telecommunication protocols anticipating alternative data paths between networks.
   - 5.1. X.25 and TCP/IP data transmission networks, protocols and devices.

6. User interface must ensure national (Latvian) language and conform to approved standard.
   6.1. 8-bit code page standard with Latvian letters LVS 8-92.
   6.2. Code page MS DOS standard RST 1040-90 must be gradually exempted from use.
   6.3. User interface must be implemented by means of 4GL in conformance with CUA.
7. Information security and user audit must conform to C2 level related to requirements of US. National Security Centre. Transmission of critical data must include public key encryption.
8. Software systems must be portable to different hardware platforms, including minicomputers and personal computers.
   8.1. Portable operating systems UNIX and Windows NT, programming environments ORACLE, Sybase, Informix-Online, ANSI C++.
9. Automated programming tools must be used in system design ensuring teamwork, documentation, readability and availability of project information, maintenance and inheritance.
   9.1. ORACLE CASE, SWH system GRAPES/GRADE or other equivalent CASE tools.
10. High quality, reliable, easily changeable and upgradeable computers must be purchased in order to achieve easy and comfortable maintenance that does not require technical personnel.
   10.1. Modular, upgradeable and enhanceable computer devices that consist of standardised components and can be configured by a system operator or by user him/herself without preliminary special training.
11. More than a year warranty must be demanded as witness of hardware quality and stable operation.
   11.1. 2-3 years warranty for personal computers and 3-5 years warranty for minicomputers.
12. Every computer program must be licensed.
   12.1. Availability of manuals, suppliers' service, joining of the Republic of Latvia to copyright conventions.
13. Workstations must be decentralised, maximally near to sources of data and users.
   13.1. Workstations and terminals at user workplaces integrated into united system.
14. Only codificators approved on national level must be used as unique keys to identify informational units.
   14.1. Personal identification code in Population register, enterprise identification code in Enterprises register, land unit code in Land and equities register, etc.
15. Data interchange must be performed using definite file formats.
   15.1. ASCII and MS DOS formats for document files, DBF format for database files, DXF format for vector graphic files and TIFF format for raster graphic files.

National Information Systems Agency (VISA)

4. National Agency for information systems is needed to implement the mentioned Directives into operation of governmental and municipal bodies, and for international representation of Latvia.
5. The Agency:
   6.1. Ensures implementation of the Directives as well as other government decisions.
   6.2. Organises suppliers' contest.
   6.3. Prepares integrated strategy for national information system development.
   6.4. Performs co-ordination and information servicing of national information systems.
   6.5. Monitors development of standards and codificators and their observation in national information systems.
   6.6. Supports development of draft laws in IT.
7. The Agency is allowed to establish an Experts Board in IT. The experts Board comprises representatives of government bodies, persons authorised by the Prime-Minister, other experts.
8. The Experts Board evaluates projects to be chosen for implementation on contest base and submits its assessments to the government, determines priorities in IT development.
9. Every implementation of IT project and technology supply for governmental and municipal bodies must be performed on contest base and in agreement with the Agency.
10. The Agency and its Experts Board inspect financial and other terms of any IT project when its summer cost is over LVL 50 000 (app. USD 85 000) and the project is financed either from state budget or from foreign aid.

Conclusive decisions

11. Until the Agency will be established its rights and responsibilities are devoted to the State Office (Valsts Kanceleja) of the Government.

# AUTOMATIC CONSTRUCTION OF TEST SETS: PRACTICAL APPROACH

Juris Borzovs, Audris Kalniņš, Inga Medvedis

Institute of Mathematics and Computer Science
The University of Latvia
Raiņa Bulv. 29, Riga 226250, Latvia

**Abstract.** The problem of symbolic execution and test generation is considered both for sequential and concurrent programs. Practical methods for test construction for the given program path are presented.

## 1. Introduction

Computer program testing (i.e., program execution on different input values - tests) remains an essential basis of program correctness decision. It is accepted that testing is not capable of program correctness proving (except cases when program is executed on all possible input values), nevertheless, in practice, if program gives correct outputs on sufficiently large amount of tests, confidence of its correctness becomes psychologically very strong.

As test generation is rather labor-consuming and quite often rather subjective, already tens of years ago trials were performed to automate this process [1,2]. One possible approach to the solution of the problem is test generation by means of symbolic execution of program paths and the following solution of path conditions (which mainly are systems of equalities and inequalities over program input parameters) obtained by symbolic execution.

Since 70-ies rather many experimental systems have been developed on the basis of the before mentioned approach [3-11]. In the second half of the 80-ies this approach has experienced the revival in the application of testing of specifications of large program systems (especially telecommunication) [12-14,28,29].

This paper deals with automated test generation methods for sequential programs and protocol specifications written in SDL language [15,16]. In both cases symbolic execution of programs is used.

The paper consists of two major parts. Part 2 deals with sequential programs. The notion of symbolic execution is formalized here. Sequential subset of SDL (equivalent to large part of Pascal) is described and an example of program is given. Correct symbolic

execution is defined for this subset of SDL and demonstrated on the program example. A heuristic method for solving equations (path conditions) obtained as the result of symbolic execution of program path is presented, thus yielding a practical method to generate a test executing a selected program path.

In Part 3 the approach is extended to concurrent programs. Correct symbolic execution is extended to all major concurrency concepts of SDL. The method is demonstrated on a realistic example - sliding window protocol in SDL, test generation procedure based on symbolic execution is shown for selected paths. Moreover, a heuristic method is presented for path selection (according to criterion C1) based on state concept related to that used for theoretical approach to test generation [17]. The method ensures the generation of test set executing all branches for the sliding window example in a reasonable time. A more sophisticated heuristic test generation method supposed to work efficiently on comparatively large SDL systems is also outlined.

Part 2 has been written by J.Borzovs and I.Medvedis. It contains results obtained by the authors at various times [10, 21, 22]. Part 3 has been written by A.Kalniņš and it contains new results.

## 2. Symbolic execution and test generation for sequential programs

Symbolic execution of programs is a wide area per se and has various applications. In this paper we restrict ourselves to the use of symbolic execution for feasibility condition description for program paths and test generation for a path based on these conditions.

Our approach to test generation by means of symbolic execution can be applied to a class of programming languages characterized by the following properties. These are block structured procedural languages with strong typing. A typical representative of this class is Pascal together with its newest derivatives like Modula-2, Turing, etc. Some restrictions, nevertheless, are present. We exclude direct memory management (pointers and related operations), calls of external procedures and functions as black boxes (with no source text available), nondeterministic functions (like random number generators).

Languages of the considered class have common property that the main control unit is a procedure with formal input parameters and declaration part defining local variables and their types. Procedure body consists of statements (assignments, conditionals, etc.) which can be represented both in conventional textual form and in flowchart like form.

Symbolic execution can be defined by our methods for any language of the class described. To do this a special symbolic execution language correlated to the given programming language must be designed. In this paper we describe only symbolic execution of SDL - in Part 2 for its sequential subset (equivalent in fact to large Pascal subset), in Part 3 we expand this definition to concurrent aspects of SDL.

### 2.1 Formalization of Symbolic Execution of Program Path

In order to define formally symbolic execution we must describe more precisely some notions present in any programming language L of the considered language class.

1. All data types permitted in the programming language L are denoted by $T_1$, $T_2$, $T_3$,.... If the language L permits only predefined types, then the number of the types used is finite. If the language L has type defining facilities (like **Array** [1..10] **Of** Integer in Pascal), then the number of possible types is infinite. Nevertheless we assume that syntax and semantics definition of the language L determines uniquely the complete type list $T_1$, $T_2$, ... and type declaration in a program is only a way to select one of these types. The domains of types (i.e., sets containing possible values of the variables of the type) are denoted by $D_1$, $D_2$, $D_3$, ... . If the language L has type naming facilities like

```
Type Seqno = Integer;
     List  = Array[1..10] Of Seqno;
Var  Buf : List;
```

in Pascal, the corresponding ground type containing no intermediate program defined identifiers and having the same domain (**Array**[1..10] **Of** Integer for variable Buf in the example) is used to denote the type of variable in our discussions. A program independent list of possible ground types $T_1, T_2, \ldots$ can actually be defined for Pascal like languages (in a way similar to that used further for SDL subset).

2. We assume type $T_1$ to correspond to normal Boolean data type, so $D_1$ = {True,False}.

3. Every program P in the language L has a certain number n of variables. Each variable has a name and a certain type (from the list $T_1,T_2,\ldots$). If program P has variables $X_1,\ldots,X_n$, then the corresponding types are denoted by $T^P_{x_1}, T^P_{x_2}, \ldots T^P_{x_n}$ and domains by $D^P_{x_1},\ldots,D^P_{x_n}$.

4. A certain number of the program variables are input parameters, i.e., variables containing program input data. We assume the first m variables $X_1,\ldots,X_m$ to be the parameters.

5. The body of program P consists of statements, each of them having one or more exits (e.g., If-statement normally has two exits), all statements are somehow labelled. A sequence $(S_1e_1, S_2e_2,\ldots, S_ke_k)$ is called a path if $S_1$ is the first statement of the program and if exit $e_i$ of statement $S_i$ determines $S_{i+1}$ as the next statement to be executed. If statement $S_i$ has only one exit or the exit is uniquely determined by some other syntactic means, $e_i$ will not be given explicitly.

Now let us describe the symbolic execution of programs in the language L.

The <u>symbolic language</u> SL corresponding to the programming language L is defined as:

1. Many-sorted signature $\Sigma$ defined over the same (ground) types $T_1,T_2,\ldots$ used in the programming language L. A special type $T_0$ with only one value undef in its domain $D_0$ is introduced (this value is used to denote undefined variable values). The signature contains function symbols $f_1,f_2,\ldots$ and for every function symbol $f_i$ its argument types and result type are specified

$$f_i: T_{i_1}, \ldots, T_{i_k} \text{ ---> } T_{i_0}$$

(including zero argument constant functions) .No function $f_i$ is defined over $T_0$, only constant function Undef has $T_0$ as value type.

2. An interpretation I of functions symbols $f_1,f_2,\ldots$ from signature $\Sigma$ .

The main objects considered in the language SL are terms. Terms in SL are well-typed expressions composed of function symbols and typed variables in normal sense. Terms are used to describe the behaviour of programs in L. Though it is not required formally, function symbols $f_1,f_2,\ldots$ and their interpretation I normally isclosely related to functions used in the language L itself or in

its semantics description.

A term in SL is said to be a predicate term if its range is $D_1$={True,False}.

We say that a term T conforms with a program P if all variables occurring in the term T are also input parameters of the program P and the type of the variable determined by its occurrence in the term T coincides with the type specified for the variable in the program P. If the language L uses type naming, then variable having some type in program P must have the corresponding ground type in term T (domains are the same!). Conformance informally means that term T is defined for the same entities which are processed by program P.

By symbolic execution (of programs in the language L) we understand an algorithm which, given a program P in the language L and a path $\alpha$ in P, produces:

1) a predicate term PC conforming with the program P,

2) for each variable $x_i$(i=1,...,n) of program P a term $T_{x_i}$ conforming with program P such that range of $T_{x_i}$ according to signature $\Sigma$ coincides with the value range of $x_i$ determined by its type (or the range is $D_0$={Undef}).

The predicate term PC is called path condition, the terms $T_{x_1},\ldots,T_{x_n}$ associated with variables - system of symbolic values, and both of them together symbolic state.

Symbolic execution is said to be correct if it produces for every program P in the language L and for every path $\alpha$ in P a symbolic state such that, for all parameter values of program P $a_1 \in D^P_{x_1}$, $a_2 \in D^P_{x_2}$,..., $a_m \in D^P_{x_m}$, there holds:

1) path condition $PC(a_1,a_2,\ldots,a_m)$=true iff the program P executed on parameter values $a_1,\ldots,a_m$ traverses the path $\alpha$,

2) if term $T_{x_i}$ is associated with variable $x_i$(i=1,..., n) according to the system of symbolic values and the value of the instantiated term $T_{x_i}(a_1,\ldots,a_m)$ is $z_{x_i}$, then after the program P has traversed the path $\alpha$ on parameter values $a_1,a_2,\ldots,a_m$, variable $x_i$ contains the same value $z_{x_i}$; if $T_{x_i}$ = Undef, then variable $x_i$ has no value assigned on path $\alpha$.

So informally path condition is an assertion on parameter values of program P in order to force the execution of this path. Path condition actually accumulates the information from the

conditional statements (If, Case,...) traversed in the path . Some assertions to prevent from overflow-like errors are also accumulated in path condition. In test generation applications path condition is used to find parameter values (i.e., a test) forcing the execution of the path.

To summarize the above mentioned we can say that symbolic execution definition for some programming language L requires three tasks to be done:

1. symbolic language corresponding to L must be defined,
2. symbolic execution algorithm must be constructed,
3. correctness of symbolic execution must be proved.

In practice the majority ofmost attention usually is paid to symbolic execution algorithm, nevertheless, the two other items are important as well.

The next sections are devoted to symbolic execution definition for a certain programming language.

## 2.2 Programing Language

In this section we consider a simple sequential programming language. Constructions of the language we denote in traditional SDL [15,16] graphical form, adhering completely to SDL syntax, although many typical SDL language constructions (such as state, signal, timer...) do not occur. The considered subset of SDL functionally do not exceed the capability of Pascal language, therefore, in order to improve readability, sometimes we present translation of SDL construction in Pascal terminology.

This simple sequential programming language is used to demonstrate symbolic execution and test generation algorithms later on. We stress that the scope of the language constructions could be substantially wider from the point of view of our methods. However, we shall consider only those language constructions having been used in examples.

In this part test generation methods are demonstrated on separate SDL procedure. In SDL language the procedure has textual and graphic parts. Textual part contains the description of procedure formal parameters, types and variables. Graphic part describes data manipulations and control flow. Further we describe this language more precisely.

## Data type definitions

1. Our language has three predefined data types: Integer, Boolean and Real associated with usual operations:

```
Newtype Integer
     Literals 0,1,2,3,... ;
     Operators
          "+"   : Integer, Integer -> Integer ;
          "-"   : Integer, Integer -> Integer ;
          "mod" : Integer, Integer -> Integer ;
          "="   : Integer, Integer -> Boolean ;
          "/="  : Integer, Integer -> Boolean ;
          "<"   : Integer, Integer -> Boolean ;
          ">"   : Integer, Integer -> Boolean ;
          "<="  : Integer, Integer -> Boolean ;
          ">="  : Integer, Integer -> Boolean ;
Endnewtype Integer;
Newtype Boolean
     Literals True, False;
     Operators
          "NOT" : Boolean -> Boolean ;
          "AND" : Boolean, Boolean -> Boolean ;
          "OR"  : Boolean, Boolean -> Boolean ;
          "="   : Boolean, Boolean -> Boolean ;
          "/="  : Boolean, Boolean -> Boolean ;
Endnewtype Boolean ;
Newtype Real
     Literals ...
     Operators
          "+"   : Real, Real -> Real ;
          "-"   : Real, Real -> Real ;
          "="   : Real, Real -> Boolean ;
          "/="  : Real, Real -> Boolean ;
          "<"   : Real, Real -> Boolean ;
          ">"   : Real, Real -> Boolean ;
          "<="  : Real, Real -> Boolean ;
          ">="  : Real, Real -> Boolean ;
Endnewtype Real
```

2. Subranges of Integer type with the following declaration:

```
Syntype Mytype=Integer
     Constants First : Last ;
Endsyntype Mytype ;
```

According to SDL semantics the behaviour of the subrange type is the same as the behaviour of the Integer type with the only difference that during the assignment of a value to subrange type variable (and in some other special cases) the range check of the value is performed.

3. Enumerated type with the following declaration:

```
Newtype Mytype
     Literals Lit1, Lit2, Lit3 ... ;
Endnewtype Mytype ;
```

For these types only the following equality relations are defined:

```
"="   : Mytype, Mytype -> Boolean ;
"/="  : Mytype, Mytype -> Boolean ;
```

4. Records (structs) with fields of any type mentioned above:

```
Newtype Mytype
     Struct
        Field1  Type1;
        Field2  Type2; ...
Endnewtype Mytype
```

5. Arrays with integer subscripts and values of any type mentioned above (including structs):

```
Newtype Mytype
     Array (Type1, Type2)
Endnewtype Mytype;
```

Here Type1 - type of index, Type2 - type of value.

Ground types corresponding to the introduced SDL types will be described in Section 2.3.

### Statements of textual part

Along with type declarations textual part may also contain the following statements.

1. Procedure heading which is the first statement in every procedure specifying its name and describing its formal parameters:

```
Procedure Myproc
     Fpar
        In      Parameter1  Type1, ...
        In/Out  Parameter2  Type2, ...
```

Types Type1, Type2... must be defined outside the procedure or must be predefined.

2. Declarations of variables:

```
     DCL
        Var1  Type1,
        Var2  Type2 ... ;
```

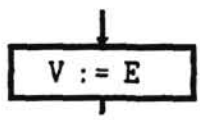
In the case of embedded procedures usual visibility rules are valid.

## Statements of graphic part

1. Procedure start:

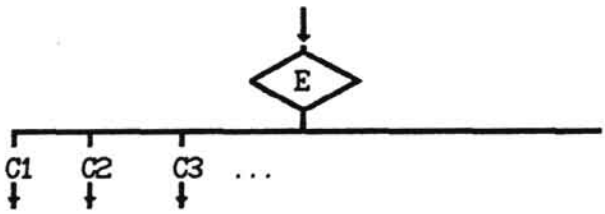
2. Procedure termination:


3. Assignment statement:



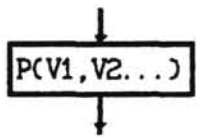

Here V is either name of variable, element of array A(I), structure element S!F or element of array of structures S(I)!F. Symbol E denotes expression of appropriate type in the usual sense.

4. Decision statement:




where E is expression of scalar type (except Real) and C1,C2,C3 ... - constants of the same type.

5. Procedure call:




where P - name of procedure; V1, V2 ... - variables or expressions of appropriate type.

## Example of Sequential Program

Let us consider a program FIND [19] which is often used to demonstrate different techniques of verification and testing.

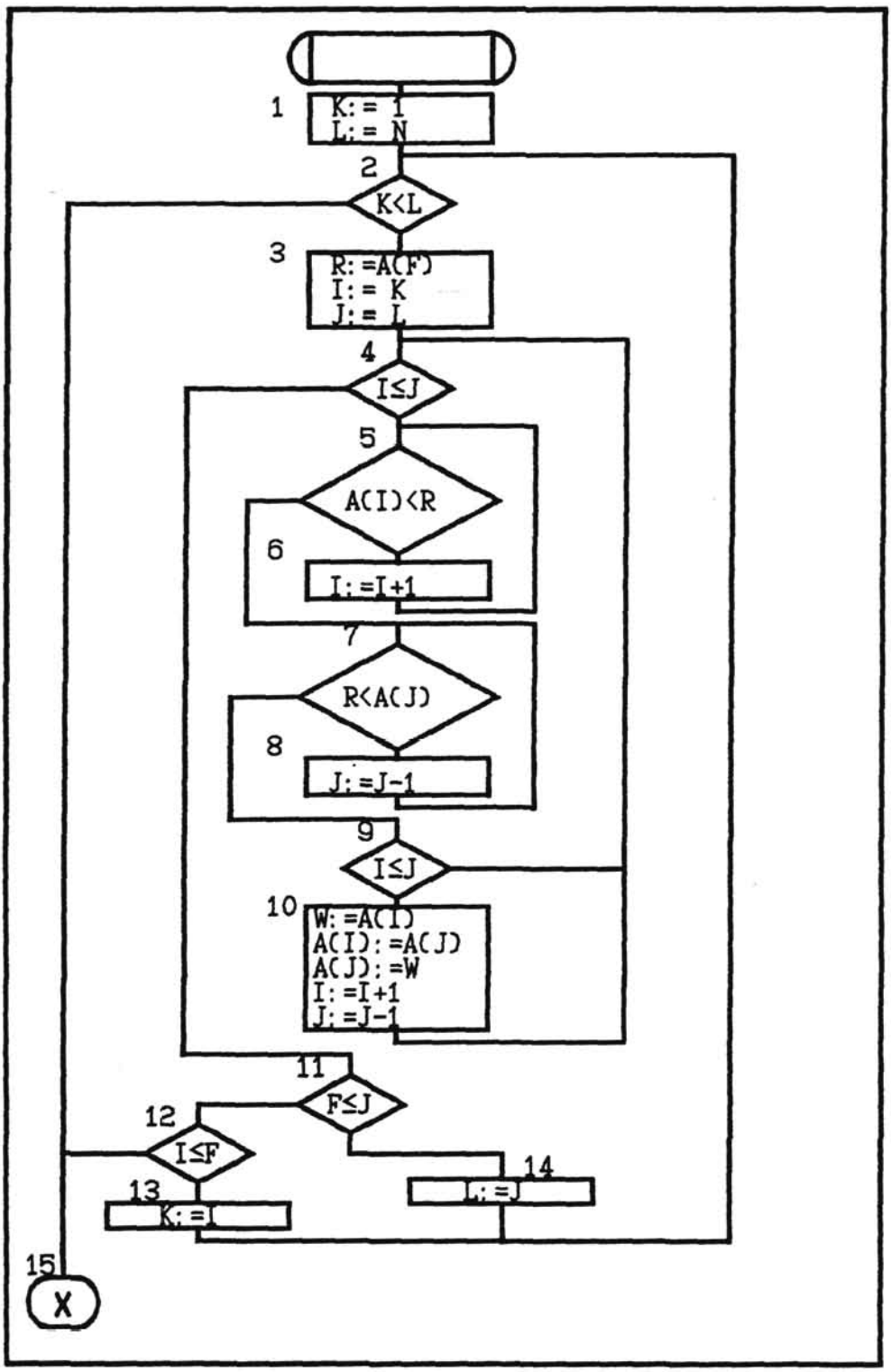
Input of the program FIND is integer array A, its length N and some integer F. The purpose of the program is to find the element of array A whose value is F-th in the order of magnitude and to rearrange the array in such a way that this element is placed in A(F) and, furthermore, all elements with subscripts lower than F have lesser values and all elements with subscripts greater than F have greater values. Thus on completion of the program the following relationship holds:

A(1), A(2), ... A(F-1) <= A(F) <= A(F+1), ... A(N)

```
Syntype Int=Integer
     Constants 1:100;
Endsyntype Int;
Newtype IA
     Array(Int,Int)
Endnewtype IA;
```

```
Procedure Find
     Fpar
        In/Out A   IA,
        In     N   Int,
               F   Int;
```

```
     Dcl
                K   Int,
                L   Int,
                I   Int,
                J   Int,
                R   Int,
                W   Int;
```


1
K:= 1
L:= N
2
K<L
3
R:=A(F)
I:= K
J:= L
4
I≤J
5
A(I)<R
6
I:=I+1
7
R<A(J)
8
J:=J-1
9
I≤J
10
W:=A(I)
A(I):=A(J)
A(J):=W
I:=I+1
J:=J-1
11
F≤J
12
I≤F
13
K:=I
14
L:=J
15
X

## 2.3 Symbolic Language

Here we describe a symbolic language *SDLS* corresponding to our SDL subset. Let us remind that we have to define many-sorted signature and its interpretation.

However, at first we must present the list of ground types for our SDL subset. Since SDL has both type defining and type naming facilities the concept of ground type is non-trivial here. So we consider all type defining constructs in SDL subset and introduce notations for the corresponding ground types.

1. Predefined types. The three predefined types: Boolean, Integer and Real are defined as ground types for all uses of these types under synonym names. For example, if type declarations

```
Syntype Counter = Integer
Endsyntype Counter;
Syntype Index = Counter
Endsyntype Index;
```

are used, the ground type corresponding to Index type is Integer.

2. All Integer subranges have the same ground type - Integer. The treatment of range checking is discussed later.

3. Enumerated types. We assume that ground type corresponding to an enumerated type determines only the number of constants, not their names. Notations for ground types are "Enumerated 1" for enumerated types with one constant, "Enumerated 2" for enumerated types with two constants, etc. For example, if we have

```
Newtype Color
     Literals Black, White ;
Endnewtype Color;
Newtype Sex
     Literals Male, Female ;
Endnewtype Sex;
```

then ground type Enumerated 2 correspond to both type Sex and type Color (Appropriate adjustment of constant functions see later).

4. Records (structs). Ground type for structs is defined as a list of ground types of struct fields. For example, if we have

```
Newtype Sex
     Literals Male, Female ;
Endnewtype Sex;
Newtype Person
     Struct
```

```
        Sex_of_Person  Sex;
        Age_of_Person  Integer ;
    Endnewtype Person;
```

then the ground type corresponding to type Person is Struct (Enumerated 2, Integer).

5. Arrays. Ground type for arrays defines the range of index and ground type of components. For example, if we have

```
    Syntype Index=Integer
        Constants 1:100 ;
    Endsyntype Index;
    Newtype Table
        Array(Index, Real)
    Endnewtype Table;
```

then the ground type of Table is Array(1:100, Real).

This completes the list of ground types (cf. $T_1, T_2, \ldots$ in 2.1). Domains corresponding to ground types follow straightforwardly from the language definition so they are discussed no more.

Now let us introduce function symbols of SDLS signature. The interpretation is described only for those functions where it is not obvious. Function symbols mainly are based on the operators introduced in our programming language (+, -, =, /=,...), and they use the same infix notation. However, most of the operators in our language are overloaded (i.e., defined for various data types simultaneously). Therefore SDLS contains derived function symbols for each of the ground types. So we have functions +, -, mod, =, /=... for Integer type; $\neg$, &, $\vee$, $=_{Boolean}$, $/=_{Boolean}$ for Boolean type and $+_{Real}$, $-_{Real}$, $=_{Real}$, $/=_{Real}$,... for Real type.

Four new groups of function symbols are introduced for complex data types (arrays and structs). They are based on functions used in SDL semantics definition.

1. Functions of extraction of value of array element $extract_T(a,i)$, where T - ground type of array as described above. These functions have array as the first argument and array index as the second one. Function yields value of corresponding array element.

2. Functions of modification of array $modify_T(a,i,v)$, where T - ground type of array, a - array, i- index of element and v- new value of element. Function yields a new array differing from the array a in the modified element i.

3. Functions of extraction of value of structure element

$extract_T(s,i)$. Functions yield the i-th element in structure s.

4. Functions of modification of structure element $modify_T(s,i,v)$. Functions yield a new structure with the value of the i-th element set to v.

In order to describe array operations adequately we assume that the first argument (array) of the function $modify_T$ can assume both values from the domain determined by type T and special value Undef of type $T_0$. (We could be completely formal in this case and introduce an auxiliary function $Undefarray_T$: $T_0$--> T, but this would make array expressions more awkward). Thus the interpretation of $modify_T$ is extended in a natural way, so that, e.g., the term
$extract_{Array(1:10,Integer)}(Modify_{Array(1:10,Integer)}(Undef,1,3),1)$,
has value 3.

The domain of modify function for structs is extended in a similar way.

The signature of SDLS also contain functions with zero arguments or constants. The same notations for constants as in SDL are used (of course, excluding overloading by means of ground type postfixes). So, e.g., the signature contains constants 1,2,3,... for Integer type, True and False for Boolean type and $1_{Real}, 2_{Real}, 3_{Real}, \ldots$ for Real type. New constant notations are introduced for enumerated types (in accordance with the corresponding ground type definitions). So the type Enumerated 2 has two constants $1_{Enumerated2}$, $2_{Enumerated\ 2}$. Constant notations adopted in SDL are used for arrays and structs, for example, $(.1,2,3.)_{Array(1:3,Integer)}$ or $(.1,18.)_{Struct(Enumerated2,Integer)}$.

This concludes the definition of the symbolic language SDLS. Unfortunately, terms in this language look very lengthy. For example, if we use data types from the program FIND, a correct term would be $extract_{Array(1:100,Integer)}(A,F)$. To make terms more readable we introduce a new notation system called the <u>derived</u> symbolic language. In this language as function and constant postfixes we do not use ground type notations but corresponding type names from the program declarations. So the beforementioned term in the derived symbolic language is $extract_{IA}(A,F)$. This is no more a correct term in the signature of SDLS, since there is no function $extract_{IA}$ in it and it can't be introduced unambiguously because identifier IA can designate various types in different programs. However, if we consider pairs <type declarations in program, term in derived language>, evidently there is an algorithm yielding an

equivalent term in SDLS for such a pair. For this reason we use the derived language without any special indication.

### 2.4 Algorithm of Symbolic Execution of Program Path

The aim of symbolic execution of program path is to obtain correct symbolic state.

Assume that a program path is given. In the case of procedure call program path contains also the corresponding sequence of statements of the called procedure. We shall show how to build symbolic state traversing the given path statement by statement.

#### Procedure Start

If symbolic execution begins with the given procedure, then variables - input parameters are assigned terms consisting of single variable, namely, the parameter itself. The rest of procedure variables are assigned undefined values (i.e., term Undef). The path condition is assigned term True.

If, on the contrary, we have reached this statement from other procedure, then, according to the range of accessibility of variable names, those pairs of variables are determined whose values are the same in the caller procedure and in this one. Formal parameters are assigned the same symbolic values as actual parameters in the caller procedure have (or terms formed by call statement). Local variables are assigned Undef values.

In both cases, if some of the input parameters are of subrange type, we also add (by means of & function) appropriate range checking predicate (such as N >= 1 & N <= 100) to the path condition.

#### Assignment Statement

The execution of assignment statement consists of value extraction of expression operands, calculation of value of expression and, the last, assignment of calculated value to the variable located in the lefthand side of the statement.

1. Extraction of values.

Due to the correspondence of names and values within the system

of symbolic values of current symbolic state we find symbolic values of variables contained in the righthand side of the statement.

If the considered variable is an element of array or record (struct), then we must form a new term using functional symbols of extract type. We also add predicates to the path condition to ensure that the range of indexes for the array is not violated.

For constants contained in the righthand side their ground types are determined (according to SDL typing rules for expressions) and the corresponding constant denotations are found.

2. Calculation of value of expression.

Taking previously obtained operand terms and functional symbols associated with corresponding operations (with correct type postfixes found) we construct a new term. If any of the operands has Undef as symbolic value, the resulting term is also Undef.

3. Assignment of value.

If the lefthand side of the statement contains scalar variable (or whole array), then the latter is assigned the newly obtained term in the system of symbolic values. If the variable is of subrange type, we add range checking predicate to the path condition, too.

If, on the contrary, the lefthand side contains array or record element, then the term of the modified value is constructed by means of modify function and this new value is assigned to the corresponding variable (i.e., array or record) in the system of symbolic values. We also enhance the path condition to ensure that the range of indexes is not violated.

## Procedure Call

Here we remind that program path contains also corresponding sequences of statements of called procedures, and we permit only calls of procedures whose texts are available.

Therefore the next statement of the path is procedure start statement of the called procedure, and the given statement will be executed when we determine the binding of actual and formal parameters of procedure statement. If actual parameter is an expression, the corresponding term is formed as in assignment statement.

### Decision Statement

As in the case of assignment statement we extract values of operands (terms) and construct the resulting term. If its type is Boolean, then such decision statement is called If statement, otherwise, Case statement.

In case of If, if the path proceeds along True-exit, the newly constructed term (simultaneously it is also a predicate term) is added to the path condition by means of & function. If the path proceeds along False-exit, we form negation of the previous term by means of ¬ function and add it to the path condition.

In the case of Case a new predicate term is constructed by the help of function = (equality) with the above mentioned term as the first argument, but the second argument is the constant assigned to the corresponding exit of the statement in the program. This last predicate term is added to the path condition.

### Procedure Exit

Local variables of the procedure are removed from the system of symbolic values.

This concludes the definition of symbolic execution. It remains to formulate the following assertion:

The symbolic execution defined in this section is correct for SDL subset described in Section 2.2.

The proof of this assertion is a little bit lengthy and is left to very patient readers.

We just note that formally correctness refers only to the basic form of symbolic language. As far as this form can be uniquely restored from the derived form and program declarations the derived form is also correct in some sense and henceforth only this form is used (sometimes omitting type qualifiers for owerloaded functions at all, if they can be uniquely restored from the context).

### Simplifier of Symbolic State

Whenever a new term is assigned to any variable or a new predicate is added to path condition we try to simplify this symbolic value or path condition. Our simplifier is rather primitive

and is mainly designed to find and calculate constant subterms in expressions. The simplifier is able to perform the following transformations:

1. Find out and calculate numerical and enumerated subterms composed of constants. Term ((1+x)+1)+1, for example, is reduced to x+3.

2. Find out and calculate subterms composed of array-type constants. Simplify array-type terms according to the following rules:

$extract_T(modify_T(A,i,x),i) \rightarrow x$

$modify_T(modify_T(A,i,x),i,y) \rightarrow modify_T(A,i,y)$

If $i \neq j$ then $extract_T(modify_T(A,i,x),j) \rightarrow extract_T(A,j)$

. . .

Here T - type of array; A - array-type term; i,j,x,y - scalar terms.

3. Simplify record-type terms the same way as arrays.

4. Reduce predicate terms to normal form. We define the normal form as conjunction $P_1 \& P_2 \& \ldots P_n$. Here $P_i$ - elementary relations in form E op F , where E and F are numerical (enumerated) type terms and op is one of the operations =, /=, >, ... Our normal form is a special case of the conjunctive normal form and, of course, arbitrary predicate term can't be reduced to such a form. Nevertheless, in order to simplify material presentation, we discuss predicate terms only in normal form.

5. Simplify elementary relations (i.e., E > F & E < F -> False).

6. Calculate constant predicate terms (i.e., P & False -> False).

It should be noted that in this section we give only examples of simplification rules, not a complete list of them.

### 2.5 Example of Symbolic Execution of Program Path

Here we do not analyze particular methods of program path selection, although one of them actually is used to select program paths to be executed (the fundamental principle is to proceed along those feasible branches having been selected less frequently; in the case of several equal variants a generator of pseudo random numbers is used).

We apply symbolic execution to the program FIND mentioned in Section 2.2, namely, to path :

(1,2,3,4,5,6,5,7,9,10,4,5,7,8,7,9,4,11,14,2,3,4,5,6,5,7,
9,10,4,11,12,15).

After symbolic execution of procedure start statement the symbolic state is as follows:

| System of symbolic values | Path condition |
|---|---|
| A = A | N >= 1 & N <= 100 & |
| N = N | F >= 1 & F <= 100 & |
| F = F | $extract_{IA}$(A,1) >= 1 & |
| K = undef | $extract_{IA}$(A,1) <= 100 & |
| L = undef | . . . |
| I = undef | $extract_{IA}$(A,100) >= 1 & |
| J = undef | $extract_{IA}$(A,100) <= 100 |
| R = undef | |
| W = undef | |

Parameters of the procedure are assigned terms consisting of single term variable, all other program variables are assigned undef term and the path condition consists of range checking predicates. Further for the sake of brevity we demonstrate only changes of symbolic state. If some term can be simplified by our simplifier, we show the result of the simplification (especially it refers to the range checking predicates).

After statement 1 (K:=1; L:=N) the symbolic state is changed as follows:

| System of symbolic values | Path condition |
|---|---|
| K = 1 | No changes |
| L = N | |

After statement 2 (If K<L true exit):

| | |
|---|---|
| No changes | 1 < N |

After statement 3 (R:=A(F); I:=K; J:=L):

| | |
|---|---|
| R = $extract_{IA}$(A,F) | No changes |
| I = 1 | |
| J = N | |

After statement 4 (I<=J true exit):

| | |
|---|---|
| no changes | no changes ( 1 <= N is reduced by simplifier) |

We conclude the example by showing the symbolic state at the end of the given path. We use the following shorthand denotation:
$B = modify_{IA}(modify_{IA}(A,2,extract_{IA}(A,N)),N,extract_{IA}(A,2))$
The resulting symbolic state is as follows:

**System of symbolic values**

$A = modify_{IA}(modify_{IA}(B,2,extract_{IA}(B,N-2)), N-2,extract_{IA}(B,2))$
$N = N$
$F = F$
$K = 1$
$L = N-2$
$I = 3$
$J = N-3$
$R = extract_{IA}(B,F)$
$W = extract_{IA}(B,2)$

**Path condition**

$extract_{IA}(A,1) < extract_{IA}(A,F)$ &
$extract_{IA}(A,2) >= extract_{IA}(A,F)$ &
$extract_{IA}(A,F) >= extract_{IA}(A,N)$ &
$4 <= N$ &
$extract_{IA}(A,3) >= extract_{IA}(B,F)$ &
$extract_{IA}(B,F) < extract_{IA}(B,N-1)$ &
$extract_{IA}(B,F) >= extract_{IA}(B,N-1)$ &
$5 > N$ &
$F <= N-2$ &
$extract_{IA}(A,1) < extract_{IA}(B,F)$ &
$extract_{IA}(A,N) >= extract_{IA}(B,F)$ &
$extract_{IA}(B,F) >= extract_{IA}(B,N-2)$ &
$F > N - 3$ &
$3 > F$

## 2.6 Method for Solving Path Conditions

In the result of the symbolic execution of program path we obtain path condition $PC(x_1...x_n)$, where $x_i$ - scalar, array or record type variable. In order to find a test case which forces this path to be executed we must solve PC as a system of equalities (inequalities). The fact that PC is reduced to normal form is irrelevant for our solution algorithm; it is used only to simplify the explanation.

Before we begin solving the path condition we, first, free it from variables and functions of record type. It can be done easily because we can assume that record fields are independent variables or arrays (if array of records). Next we separate the given path condition into independent components $PC(x_1...x_n) = P_1(x_1...x_i) \& P_2(x_{i+1}...x_j) \& ...$, where $P_k$ and $P_l$ have no common variables. After that we begin to solve these independent components.

Our method (see method of segments in [22]) is in fact exhaustive search algorithm which is improved by a number of heuristics. These heuristics are based on the study of real-life programs and are proved to be useful in test generation systems [10,22].

First let us sketch pure exhaustive search algorithm:

```
1 Procedure Resolve(P:Path_condition);
2    Select X - any variable or element of array in P;
3    For C := all possible values of X do
4        Q := P with X fixed to C;
5        Simplify Q;
6        If Q = True
7           Then System solved;
8        If Q /= True & Q /= False
9           Then Resolve(Q);
10   End
11 End Resolve;
```

To fix the value of X to C in step 4 we simply replace X by constant C or, in the case when X is an element of array (i.e., A(I)), we replace A by modify(A,I,X). In step 5 the above described simplification procedure is used to determine how successful our fixations were.

This algorithm, of course, can solve any path condition, but it is extremely impractical. After the improvements the algorithm becomes much faster, but it is not able to solve some very complex path conditions. Nevertheless, inability to solve some path condition is not dangerous for test generation system. It may lead (and even then not always) only to test systems with lower quality.

We discuss heuristics related to the three steps of the given algorithm.

First, in step 2 we select the next variable to be fixed. In practice the sequence in which we fix variables is very important for the speed-up of algorithm [8].

Second, in step 3 we try all possible values of the given variable. Yet, most of these values are not useful a priori [10].

Third, when values are fixed in step 3, first of all we must try those values that are more likely to be solutions of path condition.

Let us discuss these three heuristics separately.

## Selection of Next Variable

The following criteria are used to select the next variable to be fixed:

1. Select only scalar variables or elements of arrays that are addressed in path condition with constant index (i.e., if path condition contains extract(A,5), then we are allowed to fix A(5)). It is easy to see that this criterion can never lead us to a situation when none of the variables can be selected.

2. If criterion 1 leaves some freedom for selection, we find in path condition the elementary relation containing the least number of variables and we fix one of these variables (according to criterion 1). It allows us to simplify the path condition as early as possible.

3. If criterion 2 also leaves some freedom, we select a variable with the smallest set of admissible values (see below).

## Set of Admissible Values

With every variable in path condition we associate a set of admissible values, namely, some segment [a,b], such that no value

outside the segment can act as solution of the given path condition. We do not worry if some value inside the segment can never be solution of the system, but we are interested to keep these segments as small as possible.

For Boolean and Enumerated variables we discuss only two types of sets of admissible values: "any value (no limitations)" and "only one value C admitted". Further these sets of admissible values we call segments.

Before we begin to solve the path condition we find initial segments of variables. After fixation of every new value we revise them. First, we can set up initial segments according to the declaration of variable (for example, if the variable is a subrange). Second, a very valuable source of information is the user of the test generation system. Single remark, such as: "I am interested only in the case when all variables are less than 10", can significantly improve the performance. Third, the source of initial segments can be input/output formats, the use of variable in some language constructs, etc.

The most interesting procedure of our algorithm is reduction of segments with respect to the path condition. The aim of this procedure is to make our segments as small as possible. We apply this procedure any time when a new value of variable is fixed (after step 5). For example, if the current path condition is x+y<z & x>2 and all variables are of integer type with equal segments [1,9], we are able to reduce the segment of x to [3,7], the segment of y to [1,5] and the segment of z to [5,9]. Reduction of segments is based on simple properties of arithmetic operations and relations. For example: "If the segment of x is [a1,a2], the segment of y is [b1,b2] and the value of x+y must be in segment [c1,c2], then the segment of x can be reduced to [max(a1,c1-b1), min(a2,c2-b1)], the segment of y can be reduced to [max(b1,c1-a2),min(b2,c2-a1)] but the value of x+y must be in the segment [max(c1,a1+b1),min(c2,a2+b2)]". Or another example: "If the segment of x is [a1,a2] and path condition contains relation x=b, then the segment of x can be reduced to [b,b]".

With iterative application of these local reductions we can propagate improvements through entire path condition. The algorithm of propagation is not quite trivial and includes some new heuristics for performance improvement, yet we do not discuss it in detail.

## Selection of Values

It is possible that even after segment reduction exhaustive search is not useful. For that reason we first try to fix some outstanding values of variable: 1) both ends of segment, 2) those values within the segment that appear in the program text as constants, 3) one arbitrary point between every two values mentioned above. These rules (like our algorithm as a whole) are very simple but they work on real-life programs.

## Example of Test Generation

Now we demonstrate how the path condition produced at the end of the previous section can be solved by our methods. It is easy to see that this path condition is only roughly simplified but we do not need a stronger simplifier because our method of segments can take into account all relations between variables.

We begin with the setup of initial segments. According to the declarations, segments of all variables (i.e., N, F and A) are set to [1,100]. The second step is the reduction of segments. During reduction the segment of N is improved to [4,4], the segment of F to [2,2] but segments of A are not significantly improved. Now we must fix a value of one of the variables. It was suggested that scalar variables must be fixed first, so we can fix N to 4 (no choice).

Next we perform the simplification and the reduction of segments once more. Then, the same way, we fix F to 2 and after just another simplification and reduction of segments get the following results:

Path condition:

$extract_{IA}(A,1) < extract_{IA}(A,2)$
$extract_{IA}(A,1) < extract_{IA}(A,4)$
$extract_{IA}(A,2) >= extract_{IA}(A,4)$
$extract_{IA}(A,3) > extract_{IA}(A,4)$

Segments:

$extract_{IA}(A,1)$ [1,98]
$extract_{IA}(A,2)$ [2,100]
$extract_{IA}(A,3)$ [3,100]
$extract_{IA}(A,4)$ [2,99]

Now one element of A is to be fixed. All elements are accessed with constant indexes and three of them have segments of equal size (i.e., 1-st, 3-rd and 4-th). Let us assume that we select the 3-rd element at random and fix it to 3 (the left end of the segment). This time the following reduction of segments is not trivial, nevertheless, it is very successful:

$extract_{I_A}(A,1)$ [1,1]
$extract_{I_A}(A,2)$ [2,100]
$extract_{I_A}(A,4)$ [2,2]

So we proceed until the system is solved. It should be noted that during the solution of this system we are never forced to step back and fix a variable repeatedly. The resulting test is as follows:

A = (.1,2,3,2.)
N = 4
F = 2

If one tries to build a test for the same path manually, he probably will get a slightly different array A = (.1,4,3,2.) and will expect the procedure FIND to exchange the 2-nd and the 4-th elements. The test we have built is not so natural but it shows a significant drawback of the procedure FIND - although input array has been already partially sorted the procedure wastes time to exchange equal elements of the array.

Our path selection method coupled with the above mentioned test generation algorithm produce the following set of tests:

| | | |
|---|---|---|
| A = (.1,2,3,2.) | A = (.1.) | A = (.1,3,2,4.) |
| N = 4 | N = 1 | N = 4 |
| F = 2 | F = 1 | F = 2 |

## 2.7 Abstract Data Types and Symbolic Execution

So far we have considered only programs with predefined data types. As we know, when new data types are introduced in SDL, their semantics is specified by means of axioms. So let some new types $t_1,t_2,\ldots$ with new operators $o_1,o_2,\ldots$ of some fixed signatures be

given. The new operators are just included in the definition of symbolic functional term. (Now the formal definition of symbolic execution language changes for program to program even for the basic form of the language). The main problem is how to cope with a widened class of terms while simplifying symbolic values and solving path conditions. So we request axioms for new types and operators to be respecified as term rewriting system (TRS) rules [23,27]. The TRS should be as good as possible - confluent and terminating.

TRS describing the new types is supplied to the simplifier. As we see from the general description of the simplifier (2.4), its basic action (simplifying arithmetic, logic, array and struct terms) could also be in fact described by means of TRS (though not always with a unique normal form, due to commutative rules). So the new and basic rules are merged together making a single TRS for both old and new types. So the simplifier tries to simplify any symbolic value of a variable using this TRS as far as possible. Boolean terms in path conditions are simplified in the same way. In this paper we limit ourselves to the case when path conditions involving new types can be simplified by means of TRS to relations containing only predefined types (and boolean True or False in the best case). So the solver is not supposed to find values of new types $t_1, t_2, \ldots$ (except for trivial cases: any value and the value which has to be equal to some constant (literal) of new type).

Conditional rules in TRS (like in OBJ2 [24,27]) are also allowed, conditions should contain equalities (or inequalities) for predefined types, in particular, integers. If types and operators are generic, corresponding rules are also considered generic.

Let us consider an example: a new type queue of integers (it is used in a more general manner in Part 3). Let it have literal qnew and operators

```
qadd:integer, queue --> queue
qfirst: queue --> integer
qrest: queue --> queue.
```

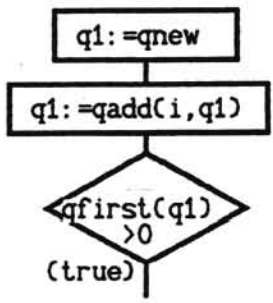
Then a standard form of TRS for this queue would be

```
qfirst (qadd (x, qnew)) --> x
qfirst (qadd (x1, qadd (x2, q))) --> qfirst (qadd (x2, q))
qrest (qnew) --> qnew
qrest (qadd (x,qnew)) --> qnew
qrest(qadd(x1,qadd(x2,q)))-->qadd(x1,qrest(qadd(x2,q)))
```

It can be simply deduced from the signatures that x, x1, x2

stand for integers, q for queue.

If we have a program fragment (with variable q1 declared as queue)




then we have at its symbolic execution:

q1=qnew (after statement1)

q1=qadd ($i_1$, qnew) (after statement2)

The true exit of the decision yields condition

qfirst (qadd($i_1$,qnew))>0

which is reduced by simplifier (using the first rule for queues) to

$i_1>0$

(a condition completely manageable by the solver).

## 3. Symbolic Execution and Test Generation for Concurrent Programs

### 3.1 General Principles of Test Generation for Communicating Processes

In this part we consider symbolic execution and automatic test generation for real time programs in the specification language SDL. Our investigations are demonstrated for a subset of SDL including all essential concepts of the language used to describe parallel processes. We consider open systems having one or more channels from environment to system ( and possibly some channels to environment). A system can contain one or more blocks, a block can contain one or more processes, procedures are also permitted. Dynamic creation of process instances is not included and all signals are assumed to be sent via channels and signalroutes. Each process is assumed to have only one instance, interprocess communication is solely by signals, viewing/revealing and export/import are not considered. We also

don't consider enabling conditions and continuous signals.

A test for an SDL system is a completely ordered sequence of input signals (including their parameter values) sent from environment to system through appropriate channels. If there are timers in the system, also the signal arrival times are fixed in the test ( if there are no timers, only the order of signals is significant).

The main goal of our research is to construct complete test set (CTS) for an SDL system. A problem of completeness criterion arises just as for sequential programs. We accept the same criterion C1, nonetheless this time its use is not so obvious, besides, its definition requires some comments. By a branch in an SDL process we understand either an input branch starting from signal input statement or a conventional program branch starting from decision statement (or START statement). A branch ends at nextstate, decision or stop statement. Criterion C1 requires every feasible branch in every process to be executed at least once. Let us remark that sometimes more stringent criteria taking into account the concurrent nature of SDL processes are used, however, there is no such one widely adopted.

All our research in SDL area is demonstrated on a popular protocol example, namely, the sliding window protocol [25,26]. At first we describe the example itself. Then we define the symbolic execution of a path in SDL system (defining at first the path itself in some reasonable way). We explain how to construct and solve path inequalities like in Sections 2.4 - 2.6. A method for constructing (potentially infinite) execution tree similar to ACT used in [14] is given. A heuristic state-based approach to construct CTS using an initial segment of this tree is briefly described, CTS for the sliding window example can be built by means of this approach. To improve the performance of CTS construction algorithms we outline the main ideas of a more sophisticated approach which uses the symbolic execution of separate processes more deeply and allows to construct CTS for this example (and even more realistic protocols) while keeping the search in reasonable limits acceptable for practice.

## 3.2 Sliding Window Protocol Example

Sliding window protocol is a popular error recovery technique used in many real protocols at data link layer. At first we present its informal description taken from [25].

### 3.2.1 Overview

The sliding window protocol supports unidirectional message flow from transmitter to receiver with positive acknowledgement sent back on each transfer. Windows are used for flow control in both transmitter and receiver. The protocol operates over a medium which may lose, reorder or corrupt messages and acknowledgements. It is assumed that corruption of messages can be reliably detected by protocol using checksums sent with messages.

### 3.2.2 Sequence Numbering

The transmitter sends a sequence number with each message. A sequence number is unbounded and is incremented for each new message. The first message transmitted is given sequence number 1.

The receiver sends an Acknowledgement when it receives a message. The Acknowledgement carries a sequence number which refers to the last message successfully transferred to the receiving user. If an Acknowledgement has to be sent before a successful reception (e.g., the first message was corrupted), it is given sequence number 0.

### 3.2.3 Transmitter Behaviour

The transmitter maintains a window of sequence numbers as shown in Figure 3.1.

This gives the lowest sequence number for which an Acknowledgement is awaited, and the highest sequence number so far used. The window size is limited to the value tws.

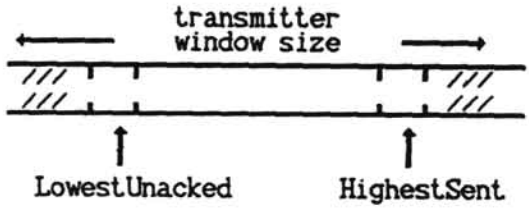



Figure 3.1. Transmitter Window Parameters

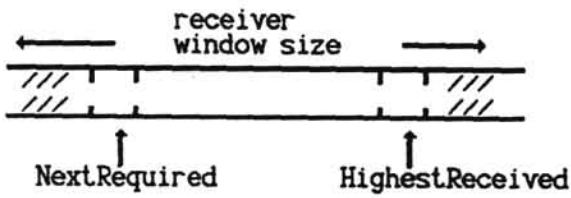



Figure 3.2. Receiver Window Parameters

The transmitter behaves initially as (a) below, and then loops doing (b), (c) and (d) where possible:

(a) LowestUnacked is set to 1 and HighestSent to 0

(b) If the current window size (HighestSent-LowestUnacked+1) is less than tws, then a message with the next sequence number (HighestSent+1) may be transmitted. In this case, HighestSent is incremented, and a timer for that message is started.

(c) If an Acknowledgement is received which is not corrupted and whose sequence number is not less than LowestUnacked, then all timers for messages up to and including that sequence number are cancelled. In this case, LowestUnacked is set to the sequence number following the acknowledged one.

(d) If a time-out occurs, then the timers for all messages transmitted after the timed-out one are cancelled. All these timed-out messages are retransmitted (in sequence, starting with the earliest) and have timers started for them.

### 3.2.4 Receiver Behaviour

The receiver maintains a window of sequence numbers as shown in Figure 3.2.

This gives the lowest sequence number which is awaited NextRequired and the highest sequence number .which has been received. The window size is limited to the value rws.

The receiver behaves initially as (a) below, and then loops doing (b) and (c) where possible.

(a) NextRequired is initialized to 1

(b) If a message is received which is not corrupted, which has not already been received and which lies within the current receive window (NextRequired +rws-1), then all messages from NextRequired up to but not including the first unreceived message are delivered to the receiving user. (There may be no such messages if there is a gap due to misordering). In this case, NextRequired is set the sequence number of the next message to be delivered to the receiving User.

(c) If a message is received under any circumstances, an Acknowledgement giving the last delivered sequence number (NextRequired-1) is returned.

### 3.2.5 SDL Description of Protocol

SDL description of the protocol is also taken from [25]. Some obvious errors are corrected and medium description is slightly changed to adapt it for testing purposes.

The description consists of three blocks representing sender, receiver and medium. Both protocol user supplying data for sender and user consuming data from receiver are located in the environment. The sending user supplies data via channel ut by signals UDTreq, the receiving one gets data via channel ur by signals UDTind. The sender forms messages from each data unit (signal MDTreq) and passes them to medium via channel mt, acknowledgements (MAKind) are received from medium via the same channel.Conversely, the receiver gets messages from medium (MDTind) and puts acknowledgements (MAKreq) onto it via bidirectional channel mr.

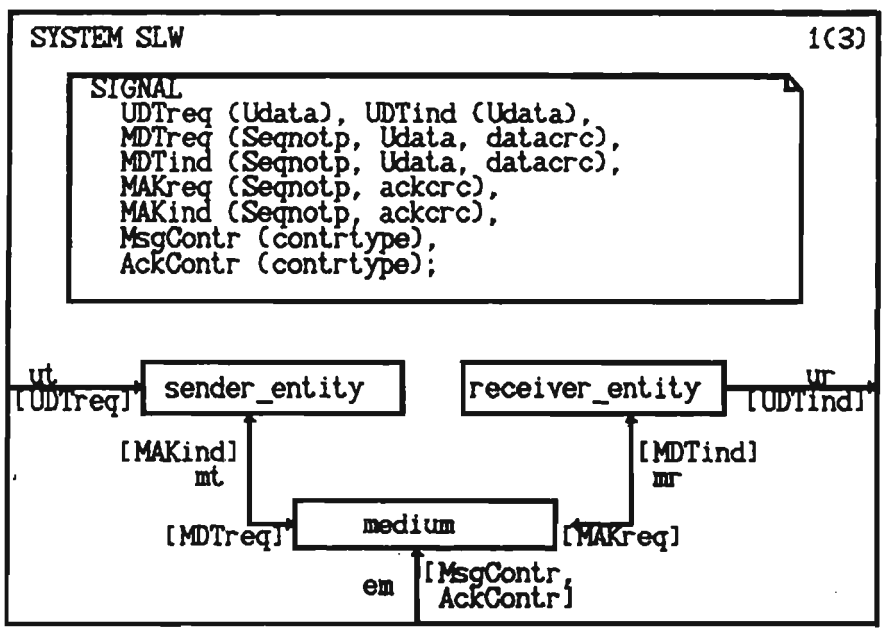
The Sender_entity block contains one process Transmit performing all sending actions. Each message (MDTreq) sent contains the generated sequence number, user data (of some unspecified type Udata) and cyclic range check computed by function dcheck. Data in transmitter window (i.e., sent but not acknowledged) are represented by queue mq, the current window limits are held in variables lu and hs. Time-out management is accomplished by setting indexed timer tim

with the corresponding seqno parameter for every message sent (and resetting it when acknowledgement arrives). The timer parameter also shows which timer instance has expired (and which messages are to be resent respectively). The time-out value is some constant delta. When the window contains maximum number of messages, the process enters the second state window_closed.

The receiver_entity block contains one process Receiver. The Next-Required sequence number is held in nr, message data received out of order (within window) are held in the array recbuf, the boolean array already_rec (of the same size) records which messages have arrived (but have not been delivered to the user yet).

The medium block contains processes MsgMan and AckMan managing the message and acknowledgement queues respectively. Message queue actions (normal transfer of message, loss of first message, reordering of messages in queue, corruption of the first message) are controlled by corresponding orders from system tester ( signal MsgContr) sent from the environment. We note that in [25] the equivalent signals are generated randomly.

In the case of normal transfer the medium actually performs only signal renaming (from MDTreq to MDTind) while retaining the same parameters. Message corruption is performed by special function corrm. Acknowledgement queue manager performs the same way.

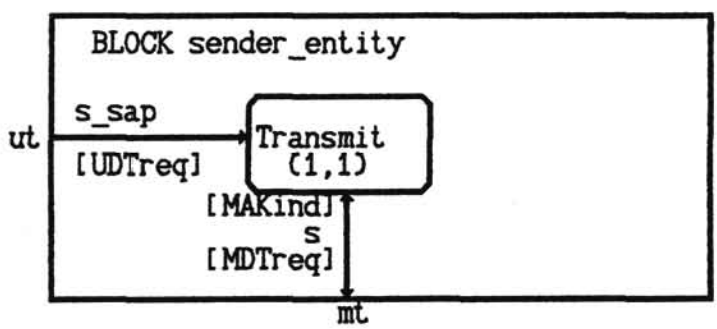


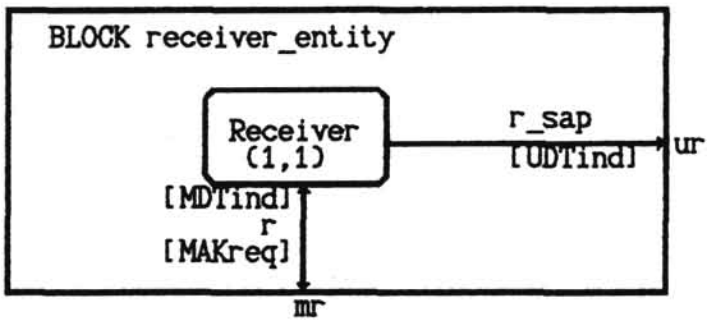
SYSTEM SLW 2(3)

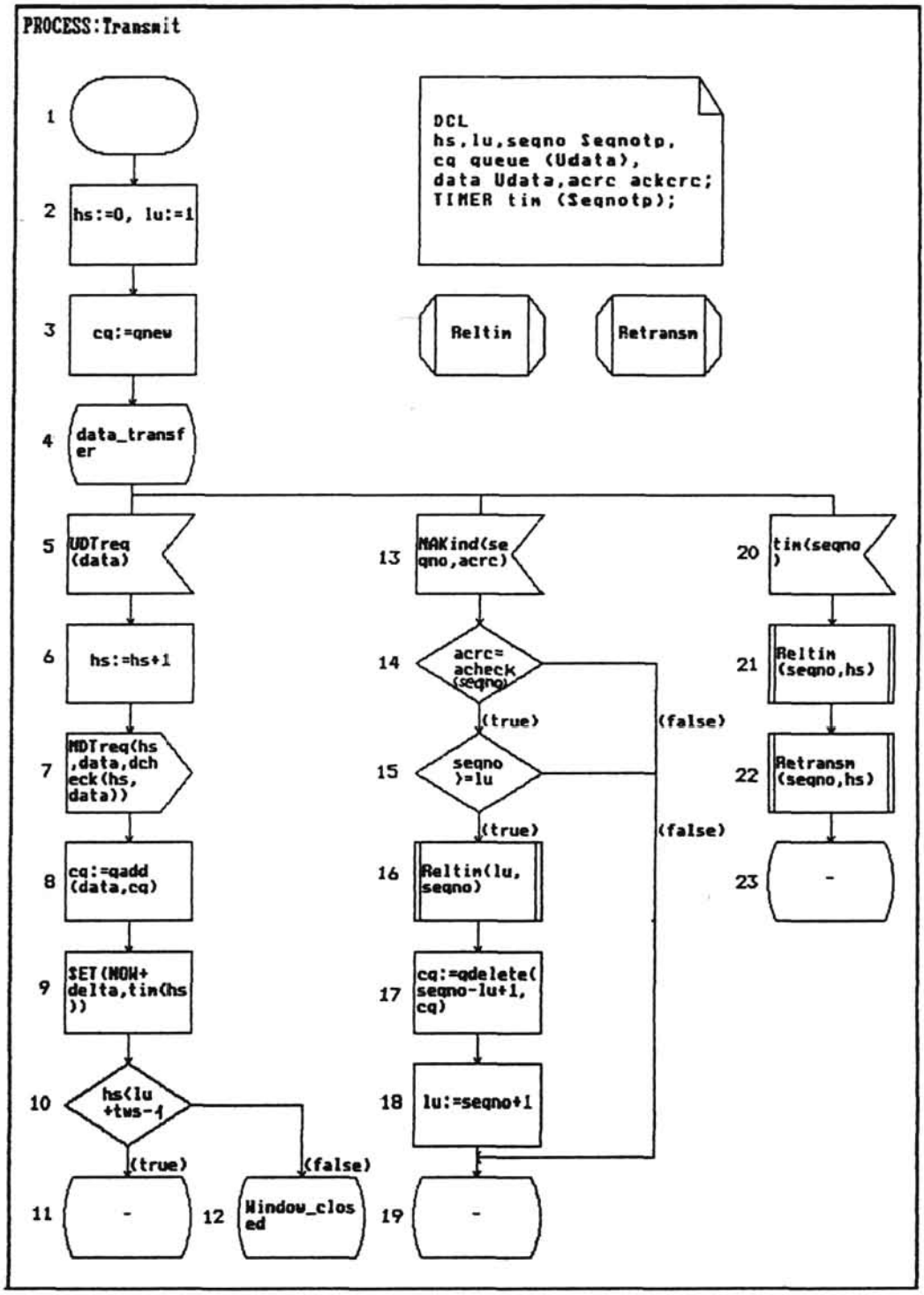
```
NEWTYPE Udata
ENDNEWTYPE Udata;

SYNTYPE Seqnotp=INTEGER
ENDSYNTYPE Seqnotp;

NEWTYPE datacrc
OPERATORS dcheck: Seqnotp, Udata ⟶ datacrc
/*builds crc field for a given pair of sequence
number and userdata in data message */
ENDNEWTYPE datacrc;

NEWTYPE ackcrc
 OPERATORS acheck: Seqnotp ⟶ ackcrc
/* builds crc field for a sequence number in ack-
nowledgement */
ENDNEWTYPE ackcrc;

NEWTYPE contrtype
  LITERALS norm, lose, reord, corr;
/* tester control options for medium action */
ENDNEWTYPE contrtype;

SYNONYM tws NATURAL = EXTERNAL;
SYNONYM rws NATURAL = EXTERNAL;
SYNONYM delta REAL  = EXTERNAL;
/* external parameters of the system */

 GENERATOR queue (TYPE item);
 LITERALS qnew;
 OPERATORS
       qadd: item, queue ⟶ queue;
      qfirst: queue ⟶ item;
      qrest: queue ⟶  queue;
      qdelete: integer,queue ⟶ queue;
      qreplace: item, queue ⟶ queue;
      qempty: queue ⟶ BOOLEAN;
 AXIOMS
 qfirst(qnew)== ERROR!;
 qfirst(qadd(x,qnew))== x;
 qfirst(qadd(x1,qadd(x2,q)))==qfirst(qadd(x2,q));
 qrest(qnew)==qnew;
 qrest(qadd(x,qnew))==qnew;
 qrest(qadd(x1,qadd(x2,q)))==qadd(x1,qrest(qadd(x2,
                                          q)));
 qempty(qnew);
 NOT(qempty(qadd(x,q)));
 qdelete(i,q)==IF i=0 THEN q
               ELSE qdelete (i-1,qrest(q))FI;
 qreplace(x1,qadd(x2,qnew))==qadd(x1,qnew);
 qreplace(x1,qadd(x2,qadd(x3,q)))==
         qadd(x2,qreplace(x1,qadd(x3,q)));
/*replaces the first element of queue by new value*/
 ENDGENERATOR queue;
```

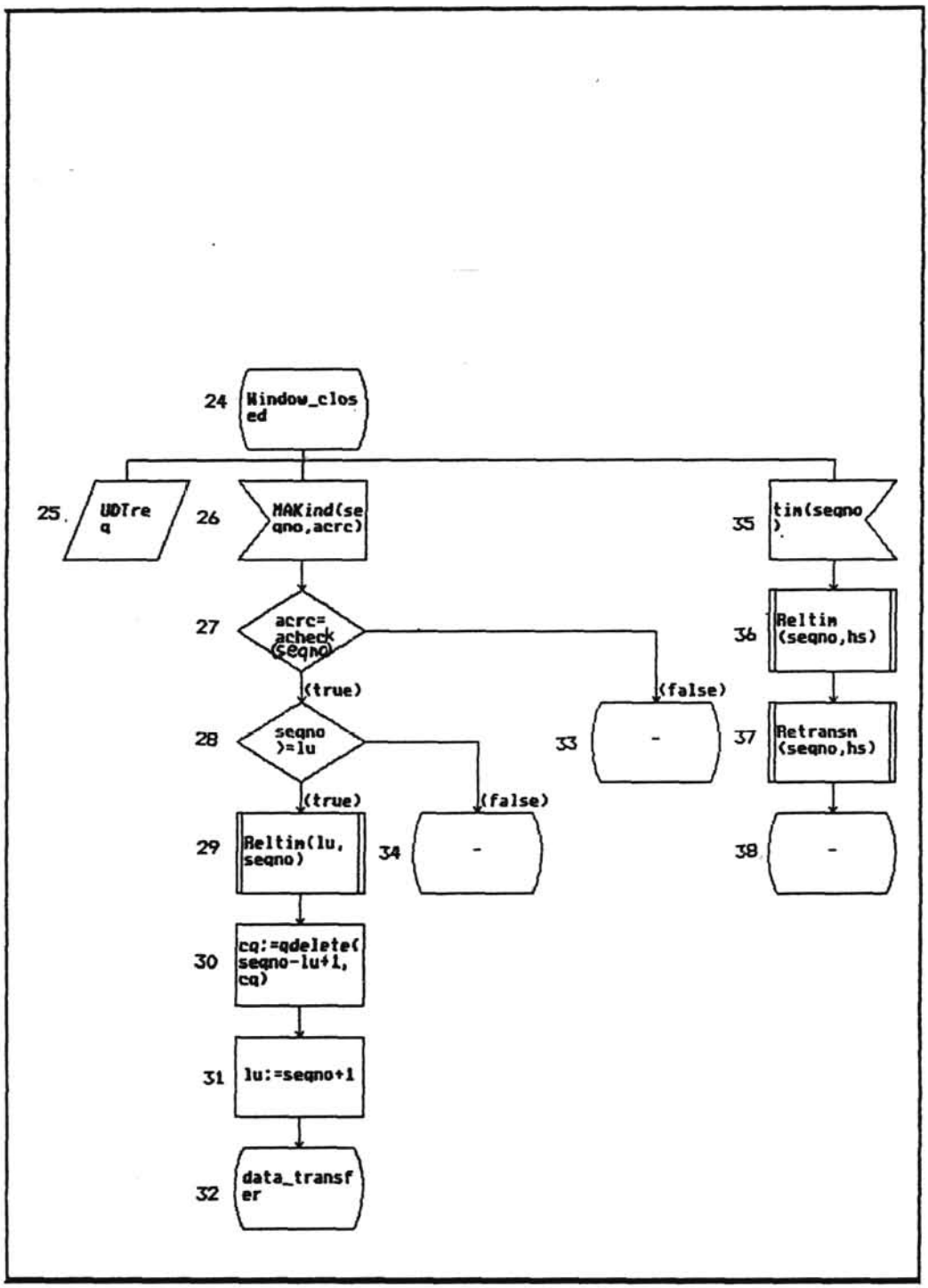
SYSTEM SLW 3(3)

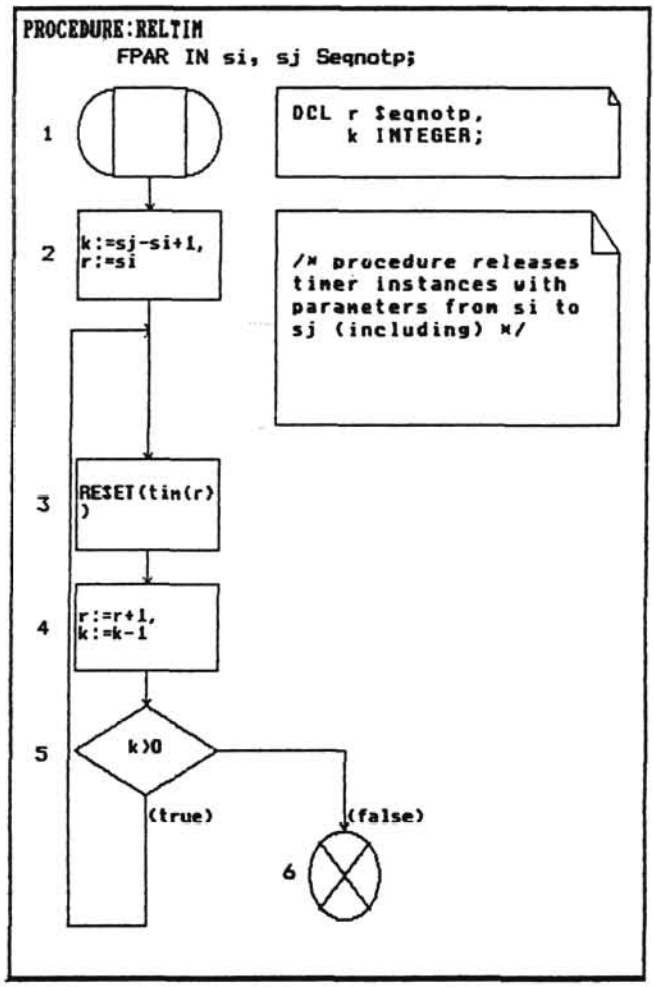
```
NEWTYPE message
      STRUCT
       seq Seqnotp;
       dat Udata;
       dc datacrc;
ADDING
OPERATORS
          corrm: message ⟶ message;
/* message corruption procedure */
  AXIOMS
   NOT (dcExtract!(corrm(m))=dcheck(seqExtract!
       (corrm(m)),datExtract!(corrm(m))));
/* every corruption is reliably detected by dcheck*/
   ENDNEWTYPE message;
NEWTYPE acknow
     STRUCT
       seq seqnotp;
       ac ackcrc;
ADDING
OPERATORS
     corra: acknow ⟶ acknow;
AXIOMS
 NOT(acExtract!(corra(a))=acheck(seqExtract!
                                  (corra(a))));
 ENDNEWTYPE acknow;
SYNTYPE rsn=INTEGER
         CONSTANTS 0:rws-1
         ENDSYNTYPE rsn;
```

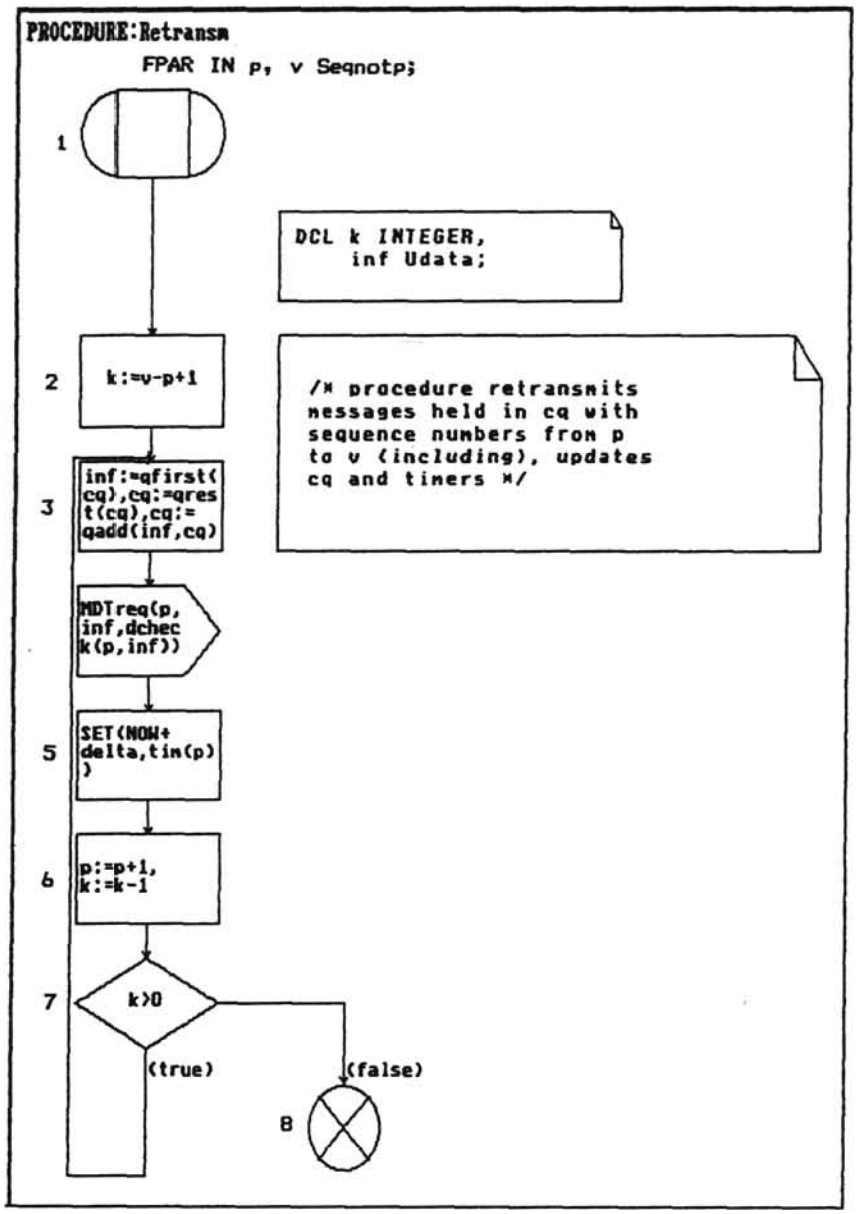


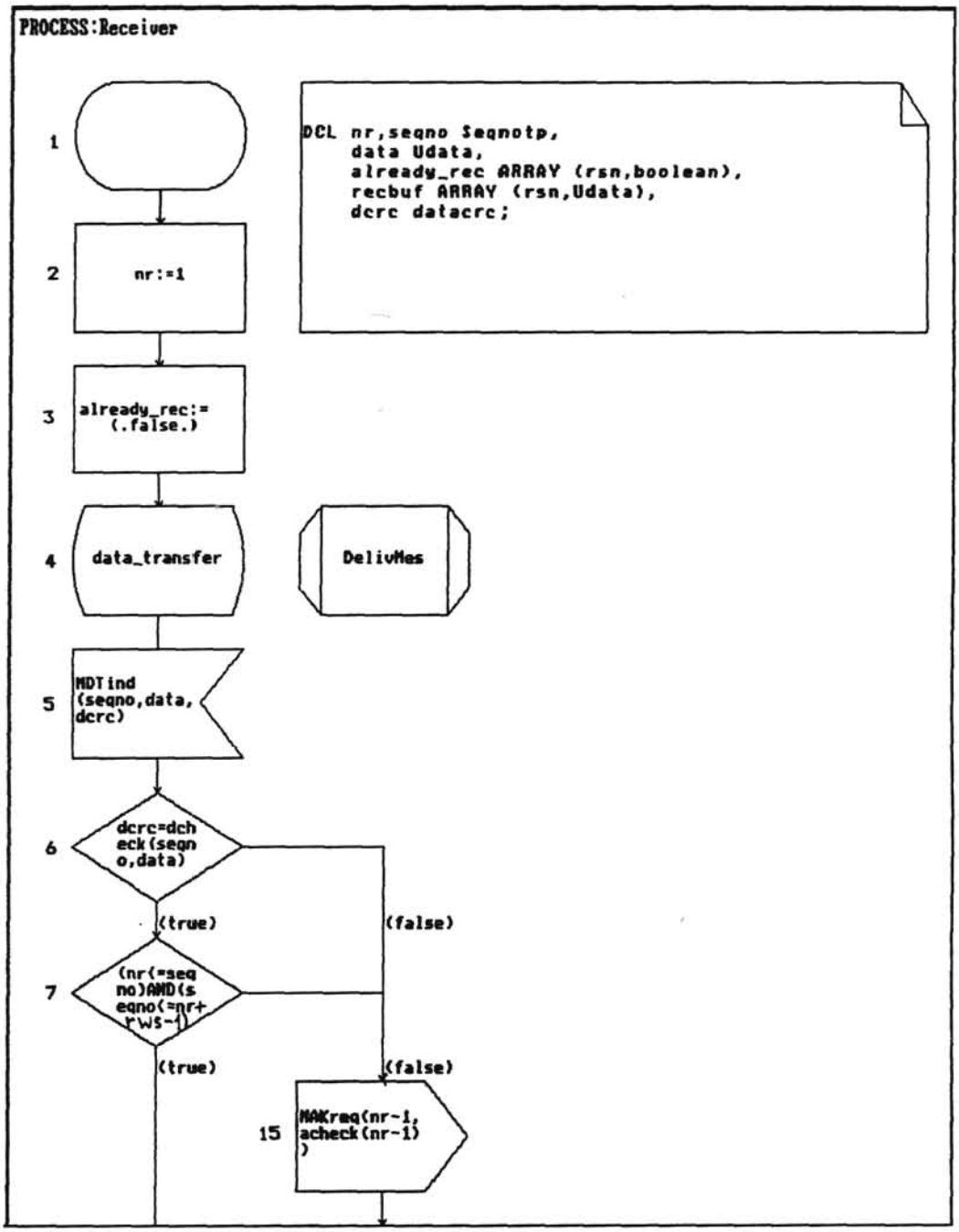

BLOCK receiver_entity
Receiver
(1,1)
r_sap
[UDTind]
ur
[MDTind]
r
[MAKreq]
mr

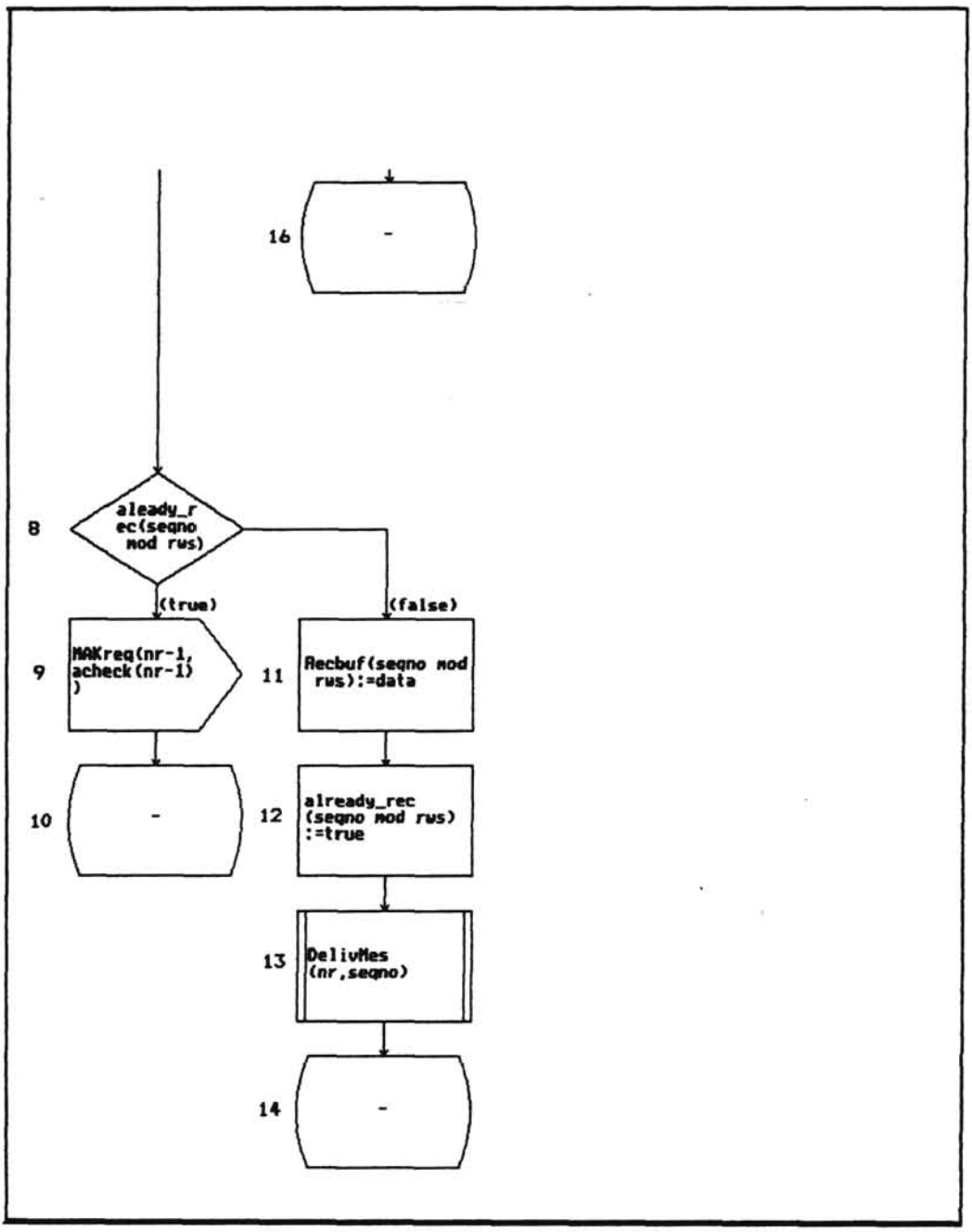

PROCESS:Transmit
1
2
hs:=0, lu:=1
3
cq:=qnew
4
data_transf er
DCL
hs,lu,seqno Seqnotp,
cq queue (Udata),
data Udata,acrc ackcrc;
TIMER tim (Seqnotp);
Reltim
Retransm
5
UDTreq (data)
6
hs:=hs+1
7
MDTreq(hs ,data,dch eck(hs, data))
8
cq:=qadd (data,cq)
9
SET(NOW+ delta,tim(hs ))
10
hs<(lu +tws-1
(true)
(false)
11
-
12
Window_clos ed
13
MAKind(se qno,acrc)
14
acrc= acheck (seqno)
(true)
(false)
15
seqno )=lu
(true)
(false)
16
Reltim(lu, seqno)
17
cq:=qdelete( seqno-lu+1, cq)
18
lu:=seqno+1
19
-
20
tim(seqno )
21
Reltim (seqno,hs)
22
Retransm (seqno,hs)
23
-

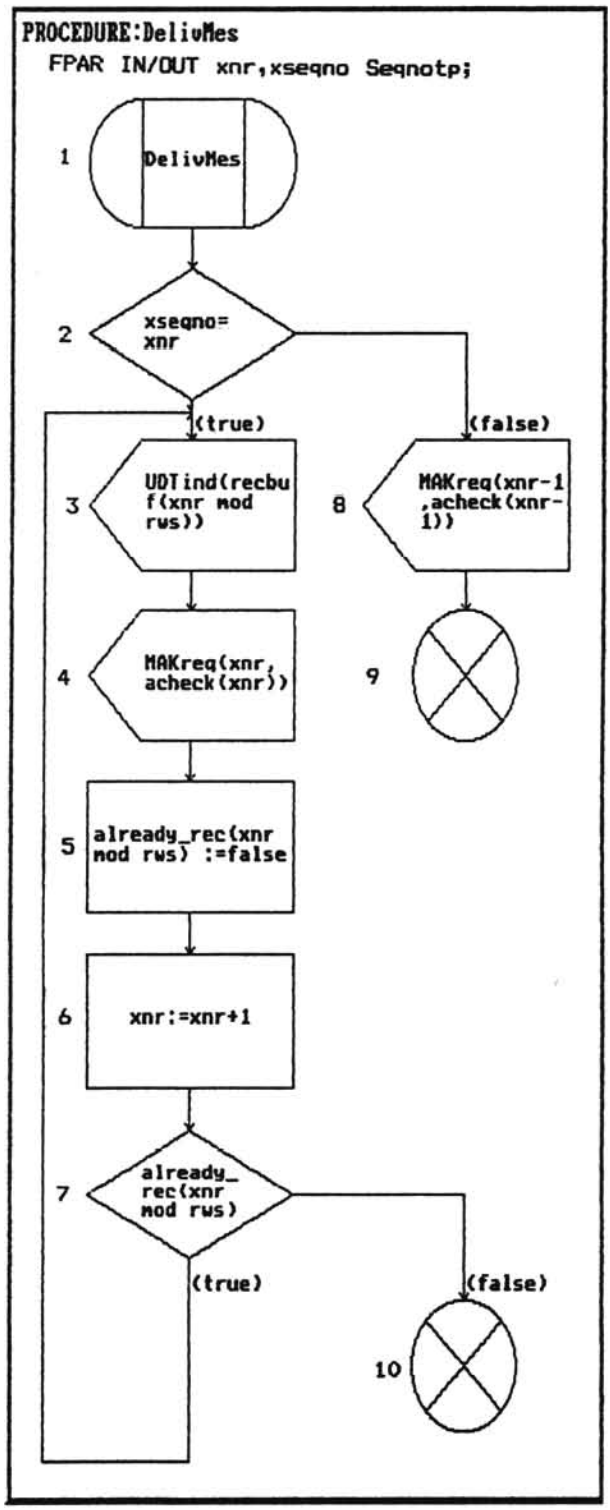

24
Window_closed
25
UDTreq
26
MAKind(seqno,acrc)
35
tin(seqno)
27
acrc=acheck(seqno)
36
Reltin(seqno,hs)
(true)
(false)
28
seqno)=lu
33
-
37
Retransn(seqno,hs)
(true)
(false)
29
Reltin(lu, seqno)
34
-
38
-
30
cq:=qdelete(seqno-lu+1, cq)
31
lu:=seqno+1
32
data_transfer

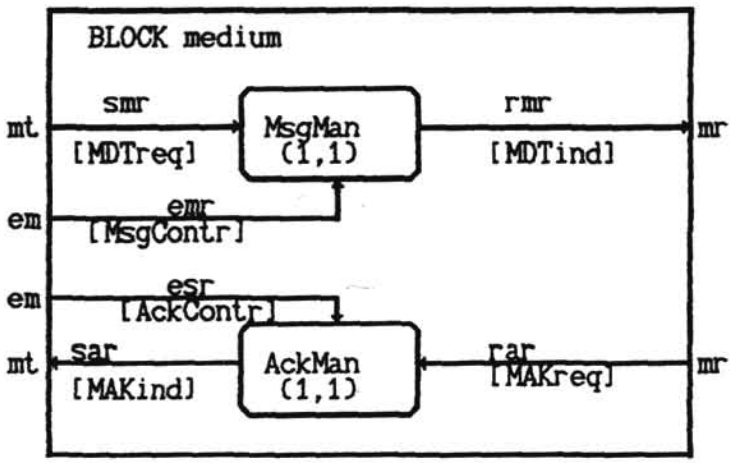

PROCEDURE:RELTIM
FPAR IN si, sj Seqnotp;
1
DCL r $eqnotp,
k INTEGER;
2
k:=sj-si+1,
r:=si
/* procedure releases
timer instances with
parameters from si to
sj (including) */
3
RESET(tin(r)
)
4
r:=r+1,
k:=k-1
5
k>0
(true)
(false)
6

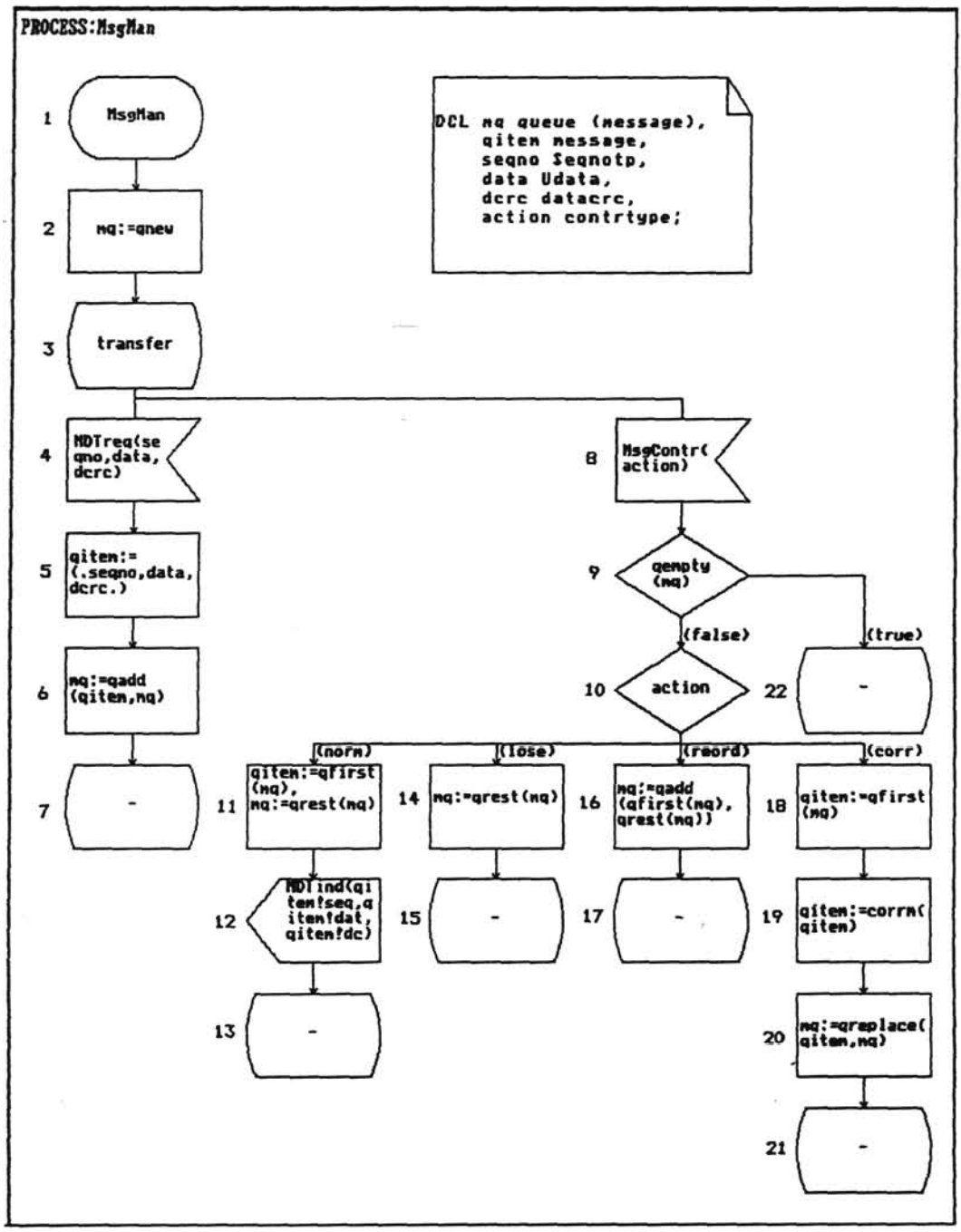

PROCEDURE:Retransm
FPAR IN p, v Seqnotp;
1
DCL k INTEGER,
inf Udata;
2
k:=v-p+1
/* procedure retransmits
messages held in cq with
sequence numbers from p
to v (including), updates
cq and timers */
3
inf:=qfirst(
cq),cq:=qres
t(cq),cq:=
qadd(inf,cq)
MDTreq(p,
inf,dchec
k(p,inf))
5
SET(NOW+
delta,tim(p)
)
6
p:=p+1,
k:=k-1
7
k>0
(true)
(false)
8

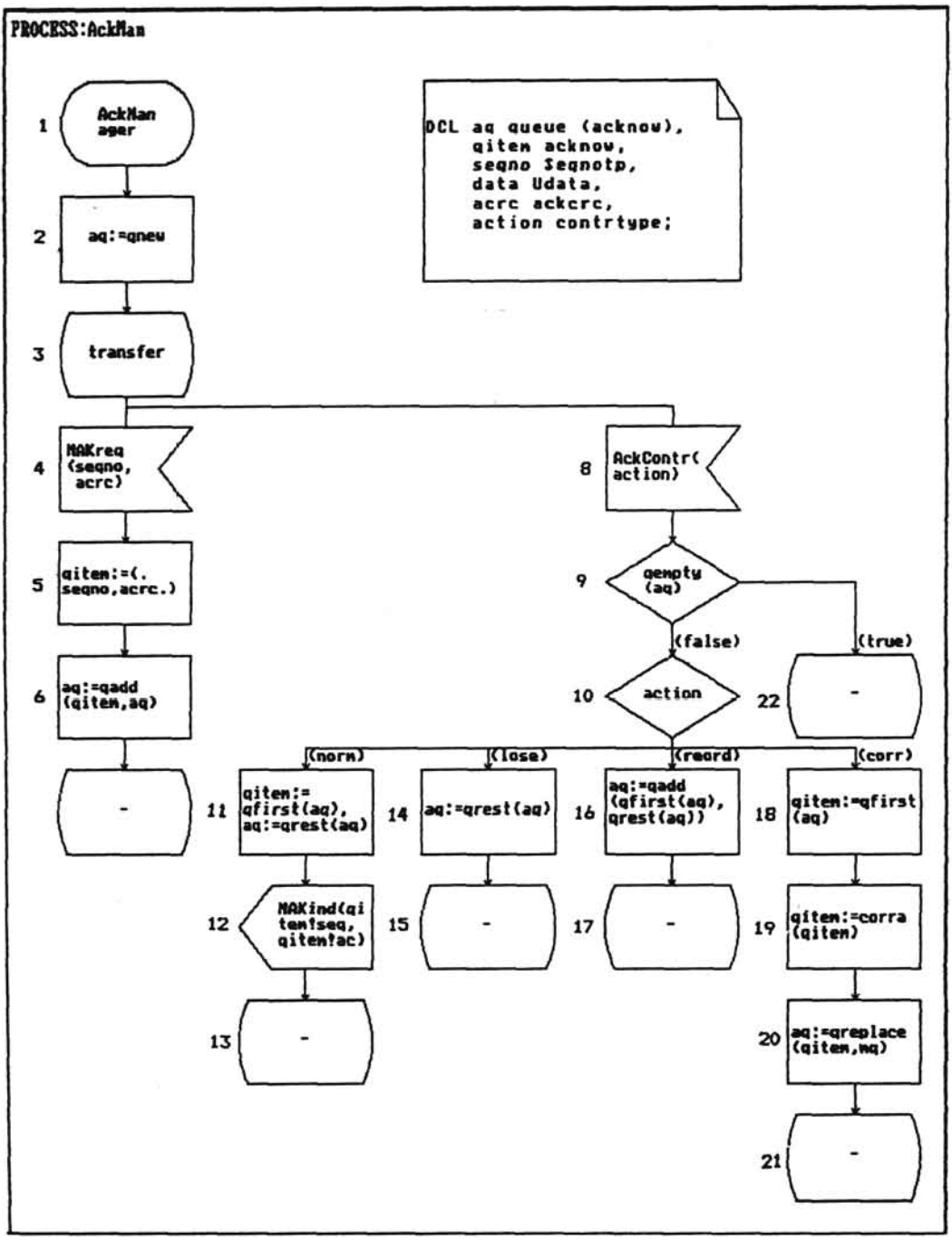

PROCESS:Receiver
1
2
nr:=1
3
already_rec:=
(.false.)
4
data_transfer
DelivMes
5
MDTind
(seqno,data,
dcrc)
6
dcrc=dch
eck(segn
o,data)
(true)
(false)
7
(nr(=seq
no)AND(s
eqno(=nr+
rws-1)
(true)
(false)
15
MAKreq(nr-1,
acheck(nr-1)
)
DCL nr,seqno Seqnotp,
data Udata,
already_rec ARRAY (rsn,boolean),
recbuf ARRAY (rsn,Udata),
dcrc datacrc;


16
-
8
already_r
ec(seqno
mod rws)
(true)
(false)
9
MAKreq(nr-1,
acheck(nr-1)
)
11
Recbuf(seqno mod
rws):=data
10
-
12
already_rec
(seqno mod rws)
:=true
13
DelivMes
(nr,seqno)
14
-


PROCEDURE:DelivMes
FPAR IN/OUT xnr,xseqno Seqnotp;
1
DelivMes
2
xseqno=
xnr
(true)
(false)
3
UDTind(recbu
f(xnr mod
rws))
8
MAKreq(xnr-1
,acheck(xnr-
1))
4
MAKreq(xnr,
acheck(xnr))
9
5
already_rec(xnr
mod rws) :=false
6
xnr:=xnr+1
7
already_
rec(xnr
mod rws)
(true)
(false)
10


BLOCK medium
smr
[MDTreq]
MsgMan
(1,1)
rmr
[MDTind]
mt
mr
em
emr
[MsgContr]
em
esr
[AckContr]
mt
sar
[MAKind]
AckMan
(1,1)
rar
[MAKreq]
mr


PROCESS:MsgMan
1
MsgMan
2
mq:=qnew
3
transfer
4
MDTreq(seqno,data,dcrc)
5
qiten:=(.seqno,data,dcrc.)
6
mq:=qadd(qiten,mq)
7
-
8
MsgContr(action)
9
qempty(mq)
(false)
(true)
10
action
22
-
(norm)
(lose)
(reord)
(corr)
11
qiten:=qfirst(mq), mq:=qrest(mq)
14
mq:=qrest(mq)
16
mq:=qadd(qfirst(mq), qrest(mq))
18
qiten:=qfirst(mq)
12
MDfind(qiten!seq,qiten!dat,qiten!dc)
15
-
17
-
19
qiten:=corrm(qiten)
13
-
20
mq:=qreplace(qiten,mq)
21
-
DCL mq queue (message),
qiten message,
seqno Seqnotp,
data Udata,
dcrc datacrc,
action contrtype;


PROCESS:AckMan
1
AckMan
ager
2
aq:=qnew
3
transfer
4
MAKreq
(seqno,
acrc)
5
qitem:=(.
seqno,acrc.)
6
aq:=qadd
(qitem,aq)
-
DCL aq queue (acknow),
qiten acknow,
seqno Seqnotp,
data Udata,
acrc ackcrc,
action contrtype;
8
AckContr(
action)
9
qempty
(aq)
(false)
(true)
10
action
22
-
(norm)
(lose)
(reord)
(corr)
11
qiten:=
qfirst(aq),
aq:=qrest(aq)
14
aq:=qrest(aq)
16
aq:=qadd
(qfirst(aq),
qrest(aq))
18
qiten:=qfirst
(aq)
12
MAKind(qi
ten!seq,
qiten!ac)
15
-
17
-
19
qiten:=corra
(qiten)
13
-
20
aq:=qreplace
(qiten,nq)
21
-

Some notes have to be added with respect to testing of the system. At first external constants (synonyms) have to be fixed. Two of them:-tws and rws are very essential, since they control loops and array sizes. Some reasonable values (not too small to make some branches infeasible, not too large to make tests enormous) are to be selected. So we set both

tws = rws = 3.

The third constant delta is less essential, it can be fixed to value, e.g., 10, when it matters.

The other problem is abstract data types. As we have explained in 2.7, axioms should be replaced by some TRS to make the simplifier work with new data types. So we present the following TRS which is "compatible" with axioms, confluent and terminating (not for every set of axioms such TRS can be found). Rules are given for the generic type queue (with some nonstandard operations) and corruption /crc check of messages (acknowledgements)

```
qfirst(qadd(x,qnew)) -->x
qfirst(qadd(x1,qadd(x2,q)))-->qfirst(qadd(x2,q))
qrest(qnew)-->qnew
qrest(qadd(x,qnew))-->qnew
qrest(qadd(x1,qadd(x2,q)))-->qadd(x1,qrest(qadd(x2,q)))
qempty(qnew)-->true
qempty(qadd(x,q))-->false
qdelete(0,q)-->q
i>0 =>qdelete(i,q)-->qdelete(i-1,qrest(q))
qreplace(x1,qadd(x2,qnew))-->qadd(x1,qnew)
qreplace(x1,qadd(x2,qadd(x3,q)))-->
          qadd(x2,qreplace(x1,qadd(x3,q)))
eq(dcExtract!(corrm(m)),dcheck(seqExtract!
       (corrm(m)),datExtract!(corrm(m))))-->false
eq(acExtract!(corra(a)),acheck(seqExtract!
                          (corra(a))))-->false
        /* eq is the equality relation */
```

(Here and further we use standard SDL syntax for struct extract functions, namely, <field_name>Extract!, not the one used in 2.3.).

### 3.3 Semantic Constraints on SDL Subset

We assume in general that SDL system is executing according to semantics of SDL-88 [15]. However, some inessential limitations and

changes are introduced to make the description of process of test generation more understandable. These changes are inessential for the example considered and, as we hope, for protocol specification in general.

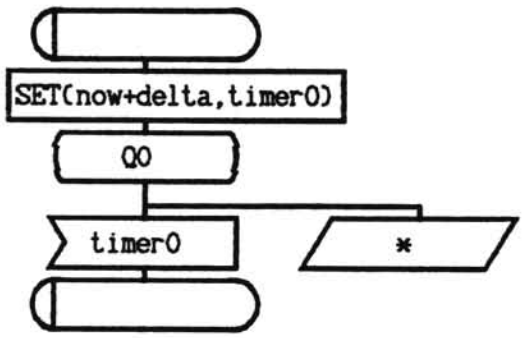
First, no two events in the whole system are assumed to be simultaneous, thus the events can be completely ordered in time. Actually, we make an even stronger assumption that only one transition from state to state occurs in the whole system at a given moment of time, the transition is always completed before another one takes place.

Second, all SDL actions including signal sending inside the system are assumed to be executing zero time. Thus, if a signal is sent from one process to other (including sending via channel), the receiving process is ready to operate just after the sending process has completed its transition, no time advancement occurs at that operation. Time is advanced only at reception of every signal from environment, and at active timer "firing".

Third, "internal" signals have priority before the signals from environment, i.e., whilst some process queue is nonempty (except the case when all existing signals are saved in the current state), no signal from environment is permitted.

The abovementioned semantic restrictions allow to assume that the whole system is executing under the control of some nondeterministic scheduler, which chooses at random an active process (i.e., a process with nonempty queue containing nonsavable signals) and activates it for one transition.If no process is active, the scheduler allows either an environment signal to arrive or an active timer to "fire" (if there is such). At the very beginning of execution the scheduler activates the initial transitions of all processes one after another.

The semantics considered is very appropriate for test generation and deterministic testing in general. To confirm practical reasonability of it we note that deterministic testing of a protocol specification makes similar assumptions as a rule [13,14]. If there are time-consuming operations in process diagrams (making zero time unrealistic), explicit timing should be introduced. We recommend delay (delta) statement for this purpose (it is in fact a macro call for the following macrodefinition:




which is completely within our SDL subset.)

Signal propagating delays along "real channels" should actually be described explicitly as testing environment controlled medium description processes in the system (e.g., MsgMan and AckMan processes in our example) in order to make delay dependencies actually testable.

### 3.4 Symbolic Execution of SDL Programs (many communicating processes)

Now, as we have discussed our semantic restrictions of SDL system behaviour, we can define the symbolic execution of a path in SDL system.

At first we have to explain what is a path in a concurrent system like SDL. We rely strongly on our semantic restrictions and the notion of the nondeterministic scheduler. Informally, a path is a particular execution trace of an SDL system. To be more formal, a path is a sequence of transition segments where each transition segment is a path from state to state (start to state, state to exit) (containing no state inside) in some process. If the path contains a procedure call, a path fragment inside the called procedure body has to follow immediately (separated into several transition segments if states are entered).If transition segments A1,A2,... referring to the same process P are singled out from path, then Ai must lead to the same state Si from which Ai+1 begins. Pseudo - transition segments (save for an environment signal, implicit transition, i.e., signal consumption in a state where it is not awaited) are also admitted in the path where they are possible according to SDL semantics. Some additional choices refining the

path will be described in the course of symbolic execution.

If the order of transition segments in the path were chosen at random, it might be highly probable that SDL semantics were violated, e.g., consumption of a signal would be required when no signal has been sent to the process. Therefore the notion of an admissible path is introduced. Informally an admissible path is one which complies with finite automata properties of SDL semantics and the scheduling principles described above, e.g., a signal can be consumed only if there is such in the corresponding queue, a timer can "fire" only after it has been set, etc. The admissibility of a path can be checked formally, but this check can be performed only along with the symbolic execution of the path. On the other hand, symbolic execution is defined only for admissible paths. We define a joint procedure for admissibility check and symbolic execution of a given path. The procedure is halted when the path is not admissible. Let us point out that admissibility does not imply feasibility, it is only a prerequisite for it.

Now let us describe the admissibility check and symbolic execution algorithm. The admissibility check is defined in the form of admissibility rules to be applied to the current symbolic state. Let an SDL system S containing processes P1,P2,...,Pn be given and $\alpha$ be a path in S. Two new "implicit" variables Q(P) and T(P) are introduced for every process P, and the symbolic values of these variables are maintained. Informally, Q(P) is the signal queue for process P, and T(P) is its active timer set. Symbolic values of Q(P) are finite sequences of symbolic values of signals denoted as <S1,...,Sk>, values of T(P) are sets of symbolic values of timers {T1,...,Te}.

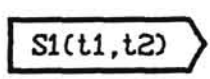
Let us begin with the description of admissibility check and symbolic execution for SDL systems without timers (as we have mentioned before, the symbolic execution and test generation is simpler in that case). So the variable T(P) will not be used for a while.

Symbolic execution is performed for every SDL statement, while admissibility check is performed only at the beginning of transition segment, i.e., when interpreting its state and input (or save) statements. In the beginning of the algorithm Q(P) are empty for all P, i.e., they contain empty signal sequence < >. Let us assume that $\alpha$ contains transition segments $A_1^{'1}, A_2^{'2}, A_3^{'3}, \ldots$, from processes $Pi_1, Pi_2, Pi_3, \ldots$, respectively. For the moment we are interested in

initial paths only, so the first admissibility rule is that $A_1^1, A_2^1, \ldots, A_n^1$ must be transition segments corresponding to start transitions (up to the first state) of all processes P1,...,Pn (in an arbitrary order), no other admissibility rules are imposed on start transitions.

Now let us define symbolic execution of SDL statements within a transition segment. Symbolic value system and path conditions are defined just as for sequential subset of SDL considered in Part 2. Symbolic execution of assignment, decision, call and procedure statements is retained just the same way. Let us remind only that each process has its own variables, thus symbolic value system contains symbolic values of variables for each process. As far as there are no input parameters for an SDL system, the only initial symbolic values used are parameter values for signals entering from environment (ENV signals for short). If the i-th instance of ENV signal S with, let's say, two parameters of types T1 and T2 is received, the symbolic values of these parameters are denoted by $S_i^1$ and $S_i^2$ (their types T1 and T2 are determined uniquely by the declaration of signal S), the whole symbolic value of signal instance is denoted by $S(S_i^1, S_i^2)$.

Let us define symbolic execution of output statement, for example,




where t1 and t2 are some expressions. Let us assume that we are within transition segment in process P1 and signal S1 is sent to process P2 (signal destination can be determined statically, i.e., from declarations in our subset of SDL). At first expressions t1 and t2 are evaluated symbolically in the context of P1 (i.e., its symbolic variable values are substituted), let the resulting terms be $t_1^s$ and $t_2^s$ . Then the symbolic value of signal $S1(t_1^s, t_2^s)$ is constructed. This value is added to the right end of signal sequence contained in Q(P2). For example, now the symbolic value of Q(P2) might be $\langle S1(t_3^s, t_4^s), S3, S1(t_1^s, t_2^s)\rangle$, i.e., the signal queue is coded in a normal way, the end of the queue being on the right.

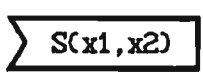
The state (better to say, next state) statement concluding the transition segment has no effect on any symbolic value. There can be no other SDL specific statements within transition segment (except timer management postponed until later and input statement in the

beginning of transition, to be considered next).

Now let us consider state and input (and save) statements. At first let us describe the admissibility rules. It follows from our scheduling principle that there can be two different situations in the whole system, namely, the one when queues Q(Pi) are nonempty for some of the processes $P_i$ (and contain signals not saved in the current state) and the other when all queues Q(Pi) are empty (or contain only saved signals). Let us consider the first situation (let us call it "internal input"). If the current transition segment refers to process P1 and starts with state ST1 followed by input of S2, then admissibility rules require the signal S2 to be in the queue Q(P1) ready for consumption. More precisely it means that symbolic value of Q(P1) either has S2 as its first element (i.e., it is of the form < $S2(t^*)$, S3, ...>) or contains S2 preceded only by signals saved in the state ST1 (i.e., the value of Q(P1) is, e.g., <S3,S4,$S2(t^*)$> where there is "save S3,S4" statement at state ST1). Implicit transition (or, in other words, the deletion of signal) is permitted when the signal in Q(P1) ready to be consumed is neither on input nor save list at state ST1. Let us note also that if several processes have signals in their queues ready for consumption, anyone of them can be selected for the current transition segment (our scheduler was assumed to be nondeterministic). No transition involving ENV signals is permitted in this situation.

Now, if the second situation takes place (we call it "external input"), a transition segment involving ENV signal input should follow. It can be a normal transition segment containing input of ENV signal S for some process P in state ST. There can also be a pseudo transition when ENV signal is saved in the queue Q(P) (if there is save symbol for S at state ST). Let us note that timer transitions are also treated as external input but they will be considered later.

Now let us define symbolic execution of state - input part. State statement has no effect on symbolic execution. Let us consider input statement, for example,




(in state ST in process P).

The symbolic value of the signal is taken (let it be $S(t_1^s,t_2^*)$),

and symbolic values of parameters are assigned to corresponding variables, e.g., x1 assumes symbolic value $t_1^*$. In the case of "internal input" the symbolic value of signal is obtained from the symbolic value of queue, i.e., the signal instance to be consumed is found in the corresponding signal sequence and after assignment this instance is deleted from the queue. In the case of "external input", as it was described earlier, new symbolic signal value $S(S_1^1,S_1^2)$ is generated (i is the number of instance of signal S sent from the environment). In the case of saving an ENV signal its symbolic value is added to the end of queue; implicit transition means the discarding of symbolic signal value.

Let us remark that generation of symbolic values for queues and signal consumption could be formalized by some TRS (using conditional rules), but we think this would add no clarity to our explanation.

Before proceeding to an example we note that a "very short" form of symbolic language is used to improve readability (type postfixes omitted at all, trivial path conditions from range checks not included).

Let us show an example of symbolic execution of a path in our system SLW. Although actually it is a system with timers, we ignore them for a moment (omitting the setting statement 9). To indicate a path we use numeric labels of statements preceded by the first letter of process name (T for Transmit, R for Receiver, M for MsgMan,A for AckMan). Exits of decision statements are not indicated explicitly (they can be deduced from the next statement label). So, let us consider an initial path

T1,T2,T3,T4,R1,R2,R3,R4,M1,M2,M3,A1,A2,A3,T4,T5,T6,T7,T8,
T10,T11,M3,M4,M5,M6,M7.

The presence of start transitions (T1,T2,T3,T4,...) for all processes in the beginning of the path was required by admissibility rules (certainly, the order is inessential). As initial transitions contain no statements of "genuine SDL", the symbolic execution proceeds the same way as in Part 2.

So we present the symbolic state after the path T1,T2,T3,T4,R1,R2,R3,R4,M1,M2,M3,A1,A2,A3 at once. All symbolic values are shown to be simplified as far as possible by the simplifier described in 2.4

| Transmit | Receiver | MsgMan | AckMan |
|---|---|---|---|
| hs=0 | nr=1 | mq=qnew | aq=qnew |
| lu=1 | already_rec= (.false,false,false.) | qitem=undef | qitem=undef |
| cq=qnew | Q(Receiver)=<> | action=undef | action=undef |
| seqno=undef | recbuf=undef | Q(MsgMan)=<> | Q(AckMan)=<> |
| data=undef | | | |
| acrc=undef | | | |
| Q(Transmit)=<> | | | |

Path condition: true

Some variables with undef values are not shown.

Statements T4,T5,...T11 form the first nontrivial transition segment (in the process Transmit). It conforms to admissibility rules since all queues are empty and "external input" occurs (namely, ENV signal UDTreq enters). After statement T5 the symbolic value of variable data is updated

$$\text{data} = \text{UDTreq}_1^1$$

(a new symbolic initial value has been generated, involving the first instance of UDTreq).

Statement T6 also updates one value

$$\text{hs}=1.$$

Statement T7 updates the queue value of process MsgMan, since channels and routes direct the signal MDTreq to this process

$$Q(\text{MsgMan})=<\text{MDTreq}(1,\text{UDTreq}_1^1,\text{dcheck}(1,\text{UDTreq}_1^1))>.$$

Statement T8 adds the following

$$C_q=\text{Qadd}(\text{UDTreq}_1^1,\text{qnew}).$$

Decision statement T10 adds no path condition since its value 1<1+3-1 is reduced to true by simplifier, the "true" exit implicitly assumed in the path is valid. Statement T11 closes the transition by returning to state Data-transfer.

Now let us consider the second transition segment M3,M4,M5,M6,M7. As the queue Q(MsgMan) is nonempty (the other queues being empty), this is the only transition segment permitted by admissibility rules in this situation. The statement M4 updates the values of variables mentioned in this input statement

$$\text{seqno}=1$$

$$\text{data}=\text{UDTreq}_1^1$$

$$\text{dcrc}=\text{dcheck}(1,\text{UDTreq}_1^1).$$

Statement M5 forms new struct value

$$\text{qitem}=(.1,\text{UDTreq}_1^1,\text{dcheck}(1,\text{UDTreq}_1^1)).$$

After M6 we have

mq=(qadd((.1,$UDTreq_1^1$,dcheck(1,$UDTreq_1^1$).),qnew).

The final symbolic state after the path is:

Transmit

hs=1

lu=1

cq=qadd($UDTreq_1^1$,qnew)

seqno=undef

data=$UDTreq_1^1$

acrc=undef

Q(Transmit)=<>

MsgMan

mq=qadd((.1,$UDTreq_1^1$,dcheck(1,$UDTreq_1^1$).),qnew)

qitem=(.1,$UDTreq_1^1$,dcheck(1,$UDTreq_1^1$).)

action=undef

Q(MsgMan)=<>

Receiver

nr=1

already_rec=(.false,false,false.)

recbuf=undef

Q(Receiver)=< >

AckMan

aq=qnew

qitem=undef

action=undef

Q(AckMan)=<>

The path condition remains true.

The path occurs to be both admissible and feasible.

Now let us consider the general case when timers are used. In this case the active timer set T(P) is maintained during the symbolic execution for every process P and admissibility rules for timers rely on this set. The initial value of T(P) is empty set {}. Timer instances (or, more precisely, symbolic values of timers) are added to the set by SET statements. The symbolic value of the timer consists of its name followed by the symbolic value of time moment to which the timer is set (and symbolic values of parameters, if there are such), for example, tcon($t^*$), tim($t_1^*$,1). The set T(P) contains at most one instance of each timer in the process P ( in the case of timers with parameters, one instance for each distinct value of parameters).

Admissibility rule for timers says that "timer transition" (i.e., transition starting with timer input) is permitted only in "external input" situation if the corresponding timer instance is in T(P) for process P under consideration. To define the symbolic execution of time involving statements, a new, real valued variable NOW is introduced (one for the whole system). The initial value of NOW is 0, and it contains the symbolic value of system time at every moment (as demanded by SDL semantics).

Basic "reference points" for time counting are times of arrival of ENV signals. Every instance of signal S sent from the environment has associated its symbolic arrival time value $S_i^T$ (i is the instance number just as for initial values of parameters). Values of the form $S_i^T$ (for all ENV signals) play the role of initial symbolic values for time counting. When the input of ENV signal S is executed, the symbolic value of NOW is set to $S_i^T$. The old symbolic value of NOW (i.e., before the new assignment, let us denote this value by NOWold) is used to add a new inequality

$$NOWold < S_i^T$$

to path condition. The inequality expresses the fact that according to our modifications of SDL semantics a new ENV signal cannot be simultaneous with some previous event in the system. The saving of ENV signal advances NOW in the same way.

For example, if we consider the previous example as a system with timers (in fact, it is such), then after statement T5 the value of NOW is

$$NOW=UDTreq_1^T$$

and the inequality $0<UDTreq_1^T$ is added to path condition (the previous value of NOW was the initial value 0).

Next we consider the symbolic execution of SET statement. This statement has the form SET(t,tim), where t is an expression of type real (as a rule, in the form $NOW+t_1$, $t_1$ can also be an expression of type real but often is a constant) and tim is a timer name. At first the symbolic value of t (denoted by $t^*$) is obtained. Then the symbolic value of timer $tim(t^*)$ is added to T(P), where P is the current process.

If there already is an instance of tim in T(P), the old instance is removed. For a timer with parameters the action is similar. Let us consider SET statement $SET(t, tim1(p_1))$. At first we assume that expression $p_1$ can be reduced to some constant $C_1$ (of the corresponding type) when computing its symbolic value $p_1^*$. Then

$tim1(C_1)$ acts in fact as an independent timer. We also assume instances of tim1 in T(P) having the same property that their parameters are reduced to constants. So symbolic value $tim1(t^*,C_1)$ is added to T(P), and, if there is an instance $tim1(t'^*, C_1)$ with the same constant parameter in T(P), the previous one is deleted.Let us return to our example and restore statement T9, omitted at first.

Let us remind that before T9 the value of NOW is $UDTreq_1^T$ and HS=1. Then after statement T9 we have a timer value

$$tim(UDTreq_1^T+delta,1)$$

and T(Transmit) assumes value

$$\{tim(UDTreq_1^T+delta,1)\}.$$

Now we have to consider the most general case when either the parameter value $p_1^*$ for the timer tim1 to be set cannot be reduced to constant by simplifier or T(P) already contains an instance of tim1 with non-constant parameter. Let us assume the instances of tim1 in T(P) to be

$$tim1(t_1,q_1),\ tim1(t_2,q_2),\ldots,tim1(t_k,q_k)$$

($q_i$ are symbolic values of the parameter).

In this moment path refinement is done. The following cases are possible here - either the new symbolic value of the parameter $p_1^*$ coincides with one of the existing values, say, $q_j$, or $p_1^*$ is a new value. Path refinement means an a priori choice of one of the possibilities (it is reasonable to call this choice a path refinement because admissibility of timer transitions later on the path depends on the choice). If the first case is chosen, $tim1(t^*,p_1^*)$ is added to T(P), $tim1(t_j,q_j)$ is removed, and besides that equality

$$p_1^* = q_j$$

is added to path condition. If the second case is chosen, $tim1(t^*,p_1^*)$ is added to T(P) and inequalities

$$p_1^* \neq q_1,\ldots,p_1^* \neq q_k$$

are added to path condition.

The symbolic execution of RESET statement is similar. The corresponding instance of the timer is simply removed from T(P) when the timer has no parameters or all parameters of timer instances (i.e., their symbolic values) can be reduced to a constant. In general case for timer resetting with parameters a similar path refinement is made and corresponding equalities (inequalities) are added to path condition.

The using of timers involves additional timing constraints in

path condition. So, when active timer set T(P) is nonempty for at least one of the processes P, additional inequalities are to be added to path condition at ENV signal input. The new symbolic value of NOW (namely, $S_i^T$ if signal S is consumed the i-th time) has to be less than the value of time held in any instance of active timer, so inequalities

$$S_i^T < t_j$$

are added to path condition for symbolic value of time $t_j$ held in any active timer instance in any of $T(P_1)$.

Let us consider an example. We extend the path considered in the previous example (with statement T9 reinserted) the following way:

T1,T2,T3,T4,R1,R2,R3,R4,M1,M2,M3,A1,A2,A3,T4,T5,T6,T7,T8,
T9,T10,T11,M3,M4,M5,M6,M7,T4,T5,T6,T7,T8,T9,T10,T11.

We describe completely the symbolic execution of the second occurrence of T5. We have before it

hs=1

$T(Transmit)=\{tim(UDTreq_1^T+delta,1)\}$

$NOW=UDTreq_1^T$

The execution of T5 gives

$NOW=UDTreq_2^T$

and two new inequalities in the path condition

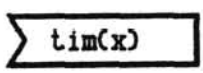
$UDTreq_1^T<UDTreq_2^T$

$UDTreq_2^T<UDTreq_1^T+delta$

After the second occurrence of T9 we have

$T(Transmit)=\{tim(UDTreq_1^T+delta,1),tim(UDTreq_2^T+delta,2)\}$

The last item to be described is the symbolic execution of timer "firing". In process P the symbolic execution of timer input, i.e., statement




,

invokes the following actions. At first an instance of timer tim is selected in T(P), let it be $tim(t^*,p^*)$ (for timers without parameters, it is the only instance of the timer, its existence is guaranteed by admissibility rules). The act of timer instance selection again is a path refinement. Then the symbolic value of NOW is set to $t^*$, the selected timer instance is excluded from T(P) and x assumes the value $p^*$. New inequalities expressing the fact that

time is nondecreasing and the timer with the least time value should "fire" the first are added to path condition. So inequalities

$$NOWold \le t^s$$

and

$$t^s \le t_j$$

for all symbolic values of time $t_j$ held in any (remaining) active timer instance in any of $T(P_1)$. Inequalities are nonstrong this time because two timers can be set on the same time moment.

Now we give an example of timer "firing" which is the continuation of the previous example with the following two transitions added:

M3,M4,M5,M6,M7,T4,T20,T21,RL1,RL2,RL3,RL4,RL5,
RL3,RL4,RL5,RL6,T22,...

(RL stands for RelTim).

The "internal input" transition M3...M7 is implied by admissibility rules (the queue Q(MsgMan) is nonempty). This occurrence of the transition is similar to the first one and affects only variables in process MsgMan, so it is not described. We start the description with T20. The previous example shows that before it there holds

$NOW=UDTreq_2^T$
$T(Transmit)=\{tim(UDTreq_1^T+delta,1),tim(UDTreq_2^T+delta,2)\}$
hs=2,

all queues are empty. So T20 is admissible, we can select the timer instance to "fire". We choose the first one. After the execution of T20 we have

$NOW=UDTreq_1^T+delta$
$T(Transmit)=\{tim(UDTreq_2^T+delta,2)\}$
seqno=1

The following inequalities are added to the path condition

$$UDTreq_2^T \le UDTreq_1^T+delta$$
$$UDTreq_1^T+delta \le UDTreq_2^T+delta$$

Just the last inequality shows that our choice of timer instances is the only possible one to obtain a feasible path (and corresponds to reasonable behaviour of timers). Had we selected the second instance, we have had contradicting inequalities in path condition

$$UDTreq_1^T < UDTreq_2^T \quad \text{and}$$
$$UDTreq_2^T+delta \le UDTreq_1^T+delta,$$

the fact obviously noticed by our inequality solver. The next

statement T22 calls the procedure RelTim, so after statements RL1,RL2 we have

k=2
r=1
si=1
sj=2,

After RL3

T(Transmit)={tim(UDTreq$_2^T$+delta,2)}

(no instance to reset actually). The path chosen in RelTim is the only feasible one in the given context, after second occurrence of RL3

T(Transmit)={ }

(because r=2 this time). So we can continue the execution, the path occurs to be feasible.

The defined timing inequalities have the property that path condition has a solution with respect to arrival times of ENV signals (i.e., the variables in the form $Si_j^T$) iff all events along the path can be allocated in time so that they comply with the details of SDL semantics laid out in Section 3.3. We could formulate this result as a theorem, had our description of symbolic execution been more formal.

We conclude this section by one more example, namely, we show the symbolic execution of another extension of the path considered in the first example. So we consider the path

T1,T2,T3,T4,R1,R2,R3,R4,M1,M2,M3,A1,A2,A3,T4,T5,
T6,T7,T8,T9,T10,T11,M3,M4,M5,M6,M7,M3,M8,M9,M10,
M11,M12,M13,R4,R5,R6,R7,R8,R11,R12,R13,D1,D2,D3,
D4,D5,D6,D7,D10,R14,A3,A4,A5,A6,A7,A3,A8,A9,A10,
A11,A12,A13,T4,T13,T14,T15,T16,RL1,RL2,RL3,RL4,RL5,
RL6,T17,T18,T19.

(Prefix D stands for procedure DelivMes, RL for RelTim).

This path corresponds to complete successful sending of one message from transmitter to receiver and successful acknowledgment sending vice versa. Active use of TRS to simplify symbolic values is demonstrated on the path. From the first example we know the symbolic state after M7:

Transmit

hs=1 T(Transmit)={tim(UDTreq$_1^T$+delta,1)}
lu=1 Q(Transmit)=< >
cq=qadd(UDTreq$_1^1$,qnew)

data=$UDTreq_1^1$
acrc=undef
seqno=undef

MsgMan

mq=qadd((.1,$UDTreq_1^1$,dcheck(1,$UDTreq_1^1$).),qnew)
qitem=(.1,$UDTreq_1^1$,dcheck(1,$UDTreq_1^1$).)
action=undef
T(MsgMan)={ }  Q(MsgMan=< >

Receiver

nr=1
already_rec=(.false,false,false.)
recbuf=undef
T(Receiver)={ }  Q(Receiver)=< >

AckMan

aq=qnew  T(AckMan)={ }
qitem=undef  Q(AckMan)=< >
action=undef

NOW=$UDTreq_1^T$

Path condition

0<$UDTreq_1^T$

External input in M3, M8 is obviously admissible.

We have after it

action = $MsgContr_1^1$

NOW = $MsgContr_1^T$ ,

path condition is augmented by

$UDTreq_1^T$ < $MsgContr_1^T$

$MsgContr_1^T$< $UDTreq_1^T$ +delta

From M9 with exit "false" we have condition

not(qempty(qadd((.1,$UDTreq_1^1$,dcheck(1,$UDTreq_1^1$).),qnew)),which is obviously reduced by a single TRS rule application to not(false)=true. So the feasibility of the selected path is not violated, no path condition is added.

Statement M10 (with exit norm implied) gives path condition

$MsgContr_1^1$ =Norm

After M11 we have

qitem= qfirst(qadd((.1,$UDTreq_1^1$,dcheck(1,$UDTreq_1^1$).),qnew))

evidently reduced by TRS to

qitem=(.1,$UDTreq_1^1$,dcheck(1,$UDTreq_1^1$).)

(namely the reduced value is fixed in symbolic value system), similarly the new value of mq is reduced to

mq=qnew.

Statement M12 augments the queue of Receiver

Q(Receiver)=<MDTind(1,$UDTreq_1^1$,dcheck(1,$UDTreq_1^1$))>

Further a nonempty queue for Receiver makes R4, R5 be the sole admissible continuation. After R5 we have in Receiver

seqno =1

data =$UDTreq_1^1$

dcrc =dcheck(1,$UDTreq_1^1$).

The exit true in statement R6 is, in fact, implied ( dcheck(1,$UDTreq_1^1$)=dcheck(1,$UDTreq_1^1$) is reduced to true by the simplifier).

The chosen exit in the next two statements is also implied, for both

(1<= 1) AND (1<=1+3-1)

and

not(extract((.false,false,false.),1 mod 3))

reduces to true.

Statements R11 and R12 make

recbuf=modify(undef,1,$UDTreq_1^1$)

already_rec=modify((.false,false,false.),1,true)).

In the procedure DelivMes we have after D1

xnr =1

xseqno=1,

which implies the chosen exit of D2 (parameters are in/out, so the changed values are returned to nr, seqno). D3 sends signal

UDTind ($UDTreq_1^1$) to environment.

We can continue the symbolic execution of the path in the same way. All remaining decisions in the path uniquely reduce to true (except one in process AckMan which gives path condition

$AckContr_1^1$ =Norm ). Admissibility rules uniquely determine the chosen internal transitions.

So we end the path with the following values (only the essential ones are given)

| Transmit | MsgMan | Receiver | AckMan |
|---|---|---|---|
| hs=1 | mq=qnew | nr=2 | aq=qnew |
| lu=2 | | already_rec= | |
| cq=qnew | | =(.false,false,false.) | |

All sets T and queues Q are empty, NOW=$AckContr_1^T$,

final path condition is

0<$UDTreq_1^T$

$UDTreq_1^T < MsgContr_1^T$

$MsgContr_1^T < UDTreq_1^T + delta$

$MsgContr_1^1 = norm$

$MsgContr_1^T < AckContr_1^T$

$AckContr_1^T < UDTreq_1^T + delta$

$AckContr_1^1 = norm.$

The path is obviously feasible. The path condition requires only the arrival times of three ENV signals to be ordered properly, taking into account also the time-out interval delta. So the following ENV signal sequence UDTreq(data1),MsgContr(norm),AckContr(norm) with arrival times 1,2,3 (if delta is assumed to be, e.g., 10) is a test executing the chosen path (value of data1 is inessential).

So it is easy to ascertain that for every feasible path in the SLW example trivially solvable path conditions can be obtained. Due to this the corresponding ENV signal sequence (with their arrival times fixed) can be generated which actually forces the execution of the path.

### 3.5 Path Selection for Test Generation - Simple Approach

As we have seen in the previous section, it is logically easy (though a little bit lengthy) to find a test (ENV signal sequence) forcing the execution of a given feasible path. In order to obtain CTS for the system SLW it would be necessary to fix some path selection strategy. However, the paths considered in the previous section show a special feature of the system SLW (and this feature is common to many protocol and similar programs). Namely, the choice of feasible path continuations in decision statements is uniquely determined (i.e., a sole exit is feasible). The only exceptions are decisions relying upon ENV signal parameters. So the only free choice is the choice of ENV signal to be received (including parameters for signals to MsgMan and AckMan). The analysis of timing inequalities show that actually two things are significant - the order of arrival of ENV signals and whether the current ENV signal arrives before the time-out period has expired ( for the timer set earliest). So the following choices are available at every "external input" point (in parenthesis the shorthand notation for the choice

is presented):

input of UDTreq (U),
input of MsgContr with parameter values norm (MN),
lose (ML), reorder (MR), corrupt (MC),
input of AckContr with parameter values norm (AN),
lose (AL), reorder (AR), corrupt (AC),
no ENV signal until the timer tim fires (T).

If we fix the ENV input string, the internal behavior of the system (and consequently, the path traversed in process bodies) is uniquely determined. As we have seen in the previous section, admissibility rules sometimes exclude T choice. The possible choices can be summarized in the following potentially infinite tree (if complete symbolic state remains unchanged after the choice, we cut off the tree after the branch). We call this tree an external signal tree (EST) (fig.3.3).

Any feasible branch in process bodies is executed somewhere in the tree. However, there is no good means to find out where exactly the point is in the tree. For example, to execute the branch R8, R9, R10, a path of length 7 in the tree is necessary (this branch seems to be the most "hidden").

As we see, the branching coefficient for the tree is 10 (excluding few first vertices), so direct exhaustive search of nearly $10^6$ vertices would not be very efficient. State based theoretical methods from [17] are not directly applicable to the example (because of potentially unlimited queues), thus some heuristic methods are necessary to limit the search.

We outline briefly one such heuristic idea which uses the state notion as in [17], however, in a more heuristic sense. We recall the notions of essential variable and essentially located statement (ELS) introduced in [10,17], Section 3.

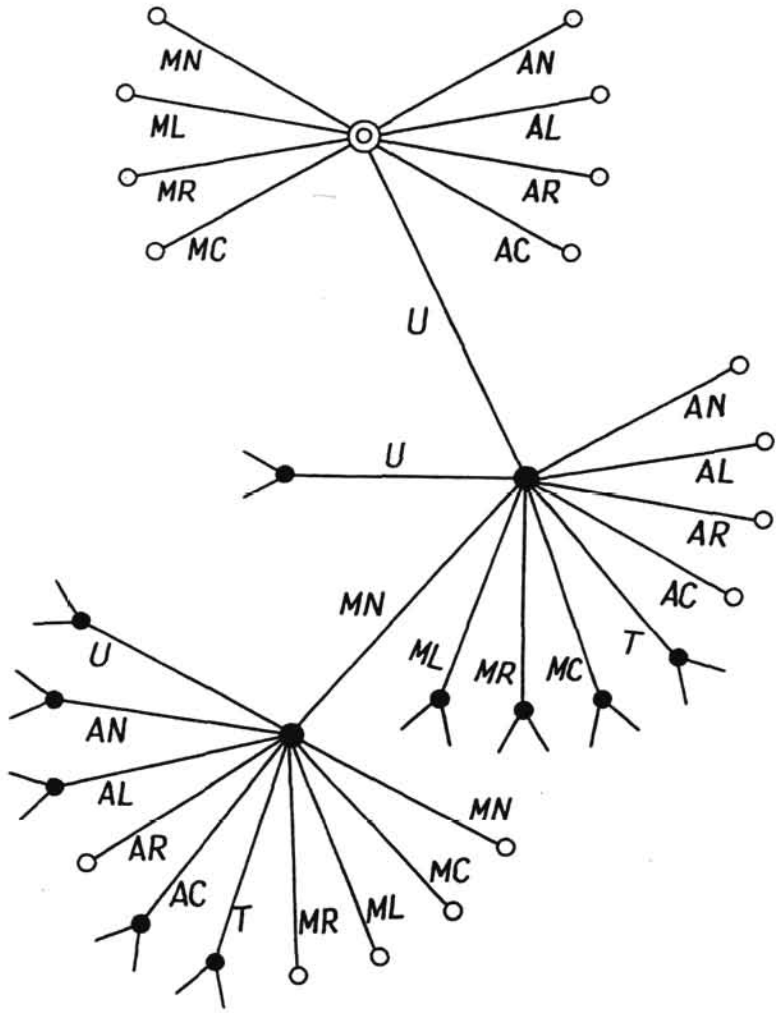



Figure 3.3. External Signal Tree

A comparatively simple analysis shows that there can be no unbounded loops within transition segments in our example. Moreover, only a bounded sequence of "internal" transition segments can follow an ENV signal input or timer firing. So we can choose the inputs of ENV and timer signals as ELS. Thus, essential variables are associated only with transitions corresponding to ENV and timer signals.

So, in a more pragmatic approach , we can say that a variable in a process is essential if it is used in decision statement and is not reassigned from ENV input to its usage in a decision (actually, for SLW example this requirement is equivalent to the formal one used in [17]. The variables affecting path admissibility are also considered essential. As we consider only external inputs as ELS, signal queues are not essential (they are always empty at these inputs), however timer sets are essential. So the following variables occur to be essential in our example: hs, lu, nr, already_rec, mq, aq, T(Transmit). Some additional arguments show that for elements of queues (mq and aq) only the sequence numbers and corrupted/not corrupted property is essential, so these elements can be reduced to pairs (s, 'n'|'c') in our state concept. For timer sets only the sequence numbers held as parameters are essential (the ordering of time moments is implied). So a reduced heuristic state containing only reduced values of essential variables is attached to each node of EST. And, as usually, the tree is cut off at state repetition, however, the finiteness of the set of states is not achieved this way. Some more stringent heuristic cut-off rules can be given, specific to this example, guaranteeing the finiteness of state set (e.g., replacing the counter values hs, lu, nr by some differences in state comparisons and estimating maximum lengths of mq, aq). A simpler heuristic approach is based on fact that all branches actually are reachable for hs, lu, nr and queue lengths not exceeding 4. So the estimated number of EST nodes to be searched for CTS building is approximately 1000 (cutting off nodes with variable values or queue lengths exceeding 4 and stopping the search when all branches have been reached). The introduced state concept actually also supports symbolic execution of each path, so the outlined approach can be used in tools generating CTS (a tool generating CTS for this example could be implemented on IBM PC). Essential variable selection and cut-off rules for states would be user supplied for

such tool. The heuristic state approach is practically acceptable for test generation for medium size protocols. A more efficient idea applicable to large systems is outlined in the next section.

To conclude the theme on signal tree we have to mention that EST is similar to asynchronous communication tree (ACT) used in [12,14]. ACT contains also signals from system to environment but our approach allows to find them as well (see signal UDTind in symbolic execution example). So the symbolic execution method allows to construct ACT also for protocols whose functioning depends essentially upon some data processing.

### 3.6 A More Intelligent Approach to Test Generation

As we have seen in the previous section, CTS can be generated on the basis of symbolic execution approach. However, though there are 34 branches (all feasible) in our example system, the state oriented approach requires considerable search (of ≈1000 states). When a human is asked to generate a test executing some branch he performs, as a rule, some backward search from the specified branch trying to find out gradually some meaningful considerations on input data (signals in our case) finally leading to some test case.

The same idea can also be used for automatic test generation. We outline it briefly on some example. So let us assume we have to generate a test executing branch D7, D3, D4, D5, D6, D7 (in procedure Delivmes in process Receiver). So for a moment we assume the process Receiver with its procedures to form a separate system (with corresponding declarations updated). So the signal MDTind is an ENV signal for the modified system, its parameters are treated as input values.

Now we try to find a feasible path containing the branch D7, D3,... . As one process is in fact a sequential program, heuristic methods from [21] can be applied. The shortest path containing the branch is R1, R2, R3, R4, R5, R6, R7, R8, R11, R12, R13, D1, D2, D3, D4, D5, D6, D7, D3,... , however, this path occurs to be infeasible (during the symbolic execution the solver founds the path condition contradictory). So by some (not described here) reasonable heuristic the next (by length) path is found, namely,

R1,R2,R3,R4,R5,R6,R7,R8,R11,R12,R13,D1,D2,D8,D9,R14,R4,R5,
R6,R7,R8,R11,R12,R13,D1,D2,D3,D4,D5,D6,D7,D3,D4,D5,D6,D7,...

The symbolic execution of the path gives the following path

condition (no timing conditions included, as there are no timers)

$MDTind_1^3$=dcheck($MDTind_1^1$, $MDTind_1^2$)

$1 \le MDTind_1^1$ & $MDTind_1^1 \le 3$

$\neg(MDTind_1^1=1)$

$MDTind_2^3$=dcheck($MDTind_2^1$,$MDTind_2^2$)

$1 \le MDTind_2^1$ & $MDTind_2^1 \le 3$

extract(modify((.false,false,false.),$MDTind_1^1$mod3,true),

$MDTind_2^1$mod3)=false

$MDTind_2^1=1$

extract(modify(modify(modify((.false,false,false.),

$MDTind_1^1$mod3,true),$MDTind_2^1$mod3,true),1,false),2)=true

The solver is able to find from the path condition unique values for numeric parameters of signals: $MDTind_1^1=2$, $MDTind_2^1=1$. The values of two other parameters are bound only by the conditions

$MDTind_1^3$=dcheck($MDTind_1^1$, $MDTind_1^2$) (*)

$MDTind_2^3$=dcheck($MDTind_2^1$, $MDTind_2^2$)

These two conditions can be treated as preconditions for the process Receiver. Thus our solver can be extended so that it can be used not only for finding test values but also for generating preconditions from path conditions (we can also consider this process as a special kind of simplification).

Now we return to the whole system and find out (statically from declarations) that signal MDTind can come only from process MsgMan. Then we consider MsgMan alone in a similar manner with both MDTreq and MsgContr treated as ENV signals. However, this time the aim is different, namely, we have to find a path in MsgMan with the specified postcondition, namely, two instances of MDTind are sent with fixed values of the first parameter 2 and 1 respectively, in addition parameters are bound by (*). Using similar heuristics for path finding, the shortest path is found

M1,M2,M3,M4,M5,M6,M7,M3,M4,M5,M6,M7,M3,M8,M9,M10,M11,M12,
M13,M3,M8,M9,M10,M11,M12,M13,

which satisfies the given postcondition.

The postcondition and path condition from symbolic execution of the path together yield:

$MDTreq_1^1=2$

$MDTreq_2^1=1$

$MDTreq_1^3$=dcheck($MDTreq_1^1$, $MDTreq_1^2$)

$MDTreq_2^3$=dcheck($MDTreq_2^1$, $MDTreq_2^2$)

$MsgContr_1^1$=norm

$MsgContr_2^1$=norm

(see how postconditions are transformed by symbolic execution into preconditions). The order of ENV signals is

$MDTreq_1$,$MDTreq_2$,$MsgContr_1$,$MsgContr_2$ (timing again is unessential). As far as only the last two signals are true ENV signals from the system point of view, the search has to be continued to obtain two input signals MDTreq from Transmit with the first four equalities as postconditions. However similar analysis of Transmit yields the postcondition to be unfeasible - because under no circumstances sequence MDTreq(2,...) MDTreq(1,...) can issue from Transmit. So another path in MsgMan (with another postcondition arising for Transmit) must be found conforming with its own postcondition. The next (by length) path is the path induced by input signals $MDTreq_1$, $MDTreq_2$, $MsgContr_1$(Reord), $MsgContr_2$(Norm), $MsgContr_3$(Norm). This path gives the postcondition for Transmit with the first two equations modified

$MDTreq_1^1$=1

$MDTreq_2^1$=2

(and equations three and four remaining the same).

This is a completely "acceptable" postcondition for Transmit. The corresponding path is induced by (this time true) ENV signals $UDTreq_1$, $UDTreq_2$. So the complete sequence of ENV signals for the system is

$UDTreq_1$, $UDTreq_2$, $MsgContr_1$(Reord), $MsgContr_2$(Norm),

$MsgContr_3$(Norm).

(or U, U, MR, MN, MN in terms of EST)

The complete ordering of signals is found substituting intermediate signals by ENV signal sequences generating these signals (likewise nonterminals are substituted by terminals in grammars). Timing conditions remain to be added to specify the test completely (the most stringent of them requesting that arrival times of all ENV signals are less than $UDTreq_1^T$+delta).

The search space for the method outlined is some tens of paths in the example considered (if powerful heuristics is used for path selection in one process). We also note that several branches in the "terminal" process (this time Receiver) can be searched for simultaneously, so reducing the complete search space for CTS even

more. So, similarly we can find that ENV signal sequence activating our "champion" branch R8,R9,R10 is (in terms of EST)

U, U, T, ML, MN, ML, MN

Maybe to find the latter path it would be more effective to consider at first Receiver alone and then Transmit and MsgMan together. Our estimate is that some hundreds of paths have to be considered to find CTS, a value completely acceptable for the example. So we hope a tool can be built using the approach outlined constructing CTS for pretty large protocols (and we hope also for large parts of electronic exchanges). However, such a tool would require some methods of reasoning on processes and pre/postconditions not completely formalized here.

We note only that the transformation of path postcondions to its preconditions by means of symbolic execution bears some resemblance to the methods used in program verification.

## 4 Conclusions

The results in both parts show that automatic test case generation has reached the status where practical implementations yielding acceptable results for programs of considerable size are possible. Certainly, such test generation systems would be complicated enough and will use the precise and heuristic methods described in the paper as well as some other ones. The main problem requiring some additional solutions is the path selection for traversing deeply "hidden" branches. In the theoretical approach the main principle used in path selection was state concept. Its modifications have proved their fitness also in a heuristic setting, however much work remains to be done to select appropriate heuristic state concepts for various classes of programs. One possible approach would be the attachment of formal comments by program authors to guide the automatic system in the right direction.

## Acknowledgements

The authors would like to thank Prof. Jānis Bārzdiņš for the setting of the problem and valuable suggestions. They also wish to thank their colleagues at the Software Research and Development Department for help in the preparation of the paper.

## References

1. Sauder R.L. General Test Data Generator for COBOL. - AFIPS Conference Proceedings, SJCC, 1962, pp. 317-323.

2. Hanford K.V. Automatic Generation of Test Cases. - IBM Systems Journal, 1970, vol. 9, No. 4, pp. 242-257.

3. Balzer R.M. EXDAMS - Extendable Debugging and Monitoring System. - In: Proc. 1969 SJCC, Montvale, N.Y., 1969, pp. 567-580.

4. Bārzdiņš J.M., Bičevskis J.J., Kalniņš A.A. Construction of Complete Sample Systems for Correctness Testing. - In: Mathematical Foundations of Computer Science, Berlin: Springer, 1975, pp. 1-12.

5. Howden W.E. Methodology for the Generation of Program Test Data. - IEEE Trans. Comput., vol C-24, pp. 554-559.

6. Clarke L.A. A System to Generate Test Data and Symbolically Execute Programs. - IEEE Trans. Software Eng., 1976, vol. SE-2, No. 3, pp. 215-222.

7. King J.C. Symbolic Execution and Program Testing. - CACM, 1976, vol. 19, No. 7, pp. 385-394.

8. Ramamoorthy C.V., Ho S.B.F., Chen W.T. On the Automated Generation of Program Test Data. - IEEE Trans. Software Eng., 1976, vol. SE-2, No. 4, pp. 293-300.

9. Pravilschikov P.A. Test Generation for Programs. - Avtomatika i Telemekhanika, 1977, No. 5, pp. 147-160 (In Russian).

10. Bičevskis J., Borzovs J., Straujums U., Zariņš A., Miller E.F. Jr. SMOTL - a System to Construct Samples for Data Processing Program Debugging. - IEEE Trans. Software Eng., 1979, vol. SE-5, No. 1, pp. 60-66.

11. Pozin B.A. A Method of Structural Test Generation for Programs. - Programmirovanie, 1980, No. 2, pp. 62-69 (In Russian).

12. Hogrefe D. Automatic Generation of Test Cases from SDL

Specifications. - In: SDL Newsletter, 1988, No. 12, pp. 34-52.

13. Kristoffersen F. Conformance Testing Based on SDL Specifications. - In: SDL'89: The Language at Work, North-Holland, 1989, pp. 257-266.

14. Bromstrup L., Hogrefe D. TESDL - Experience with Generating Test Cases from SDL Specifications. - In: SDL'89: The Language at Work, North-Holland, 1989, pp. 267-280.

15. CCITT : Specification and Description Language (SDL). Recommendations Z.100. - CCITT Blue Book, 1988, 199 p.

16. Saracco R., Smith J.R.W., Reed R. Telecommunication Systems Engineering Using SDL. - North-Holland, 1989, 633 p.

17. Auziņš A., Bārzdiņš J., Bičevskis J., Čerāns K., Kalniņš A. Automatic Construction of Test Sets: Theoretical Approach. - This volume.

18. Wirth N. Systematic Programming. - Prentice-Hall, 1973.

19. Hoare C.A.R. Algorithms 65; FIND. - CACM, 1961, vol 4, No. 1, p. 321.

20. Hoare C.A.R. Proof of Programm FIND. - CACM, 1971, vol. 14, No. 1, pp. 39-45.

21. Borzovs J.V., Urtāns G.B., Shimarov V.A. Program Path Selection for Test Generation. - Upravlayuschie Sistemi i Mashini, 1989, No. 6, pp. 29-36 (In Russian).

22. Borzovs J.V., Medvedis I.E., Urtans G.B. The Segment Method for the Solution of Systems of Equalities and Inequalities at Test Generation for Program Validation. - Upravlayuschie Sistemi i Mashini, 1990, No. 2, pp. 49-58 (In Russian).

23. Huet G., Oppen D. Equations and Rewrite Rules: a Survey. - In: Formal Languages: Perspectives and Open Problems, Academic Press, N.Y., 1980.

24. Futatsugi K., Goguen J.A., Jouannaud J.P., Meseguer J. Principles of OBJ'2. - In: Proceedings of Principles of Programming Languages, ACM, 1985.

25. Guidelines for the Application of Estelle, Lotos and SDL, Draft Manual. - CCITT, Geneva, 1988, 347 p.

26. Stenning N.V. A Data Transfer Protocol. - Computer Networks, 1976, No. 1, pp. 99-110.

27. Bergstra J.A., Heering J., Klint P. (ed.) Algebraic Specification. - ACM Press, N.Y., 1989, 397 p..

28. Sato F., Katseryama K., Mizuno T. TENT: Test Sequence Generation Tool for Communication Systems. - In: FORTE'89, Proceedings of 2nd Int. Conf. on Formal Description Techniques, North Holland, 1990, pp. 1-6.

29. Chan W.Y.L., Vuong S.T., Ito M.R. On Test Sequence Generation for Protocols. - In: Proceedings of the IFIP WG 6.1 Nineth Int. Workshop on Protocol Specification, Testing and Verification, 1989, North Holland, 1989.

PUBLICĒTS:

J. Bārzdiņš D. Bjørner (Eds.)

# Baltic Computer Science

Selected Papers

# Information Technology in Latvia: Laws and Standards

*Juris Borzovs, Dr. sc. comp.*
*Riga Information Technology Institute*

***The article reviews the current situation with laws and standards that are associated with IT in Latvia. The conclusion is that several areas of the law are not yet in order and do not conform to the requirements of the European Communities. Without going into detail, the author points to the most urgent tasks that must be accomplished. The article is based on a paper that was delivered at the conference "Latvia on the Road to an Information Society" in Riga in April 1996.***

## LAWS AND STANDARDS: MANDATORY OR FREE WILL?

The laws of foreign countries and international institutions such as the European Communities cannot come into effect in Latvia unless they are approved by the Latvian parliament. If Latvia is serious in its intent to join the European Communities, the legislative body will have no choice but to engage in a process to systematically bring Latvian law into concert with European Communities directives (see, e.g., [1]).

Individuals, for their part, must follow the laws of the country in which they reside. This is a mandatory process.

A different situation exists with standards. Unlike the situation in the former USSR, standards today are no more than recommendations, and people everywhere, including in Latvia, choose voluntarily whether or not they will follow the standards. No standard has the force of law behind it. The observance of a standard can be made mandatory by setting out such a requirement in a treaty or a normative act such as a law, an order by the Cabinet of Ministers, a decision by the president of a company, etc. The non-mandatory nature of standards does not mean, however, that they should be ignored from the outset. Standards usually are based on extensive accumulated experience, and they strike a balance between what is wanted and what is practically obtainable. The ignoring of such experience can end badly.

## THE CHALLENGE POSED BY IT TO SOCIETY

Information technology tears down borders. The Internet, for example, is built in such a way that users often do not know the route their postings have taken. Exaggerating only a bit, one might say that under certain circumstances, an E-mail message sent to one's next-door office may well spend time in a foreign country or even a different continent before arriving at its final destination. It has become virtually an everyday thing to wander around the vast information resources of the World Wide Web, paying no attention whatever to the question of where the various collections of information exist physically. An information world is arising that has no need for national boundaries. Differences between centers and peripheries disappear. Information technology truly "opens the world" for us.

This "open world", however, requires a certain unification of the laws and standards of various countries. National laws and standards, if they deal with something more than issues of interest to a specific territory (use of national language, for example), can be pointless or even cumbersome. The same holds true with respect to the absence of certain laws. Thus, for example [2] Italy's Fiat company had problems in transmitting personal data about their employees in France to the company headquarters, because the French government became involved in the process. The reason was France's tough laws concerning the defense of personal data and privacy interests and the absence of similar laws in Italy. The French government was justified in feeling that the privacy rights of French citizens, even if they are employees of Fiat, must not be violated unlawfully by sending data for storage in Italy.

So the world is being opened for us, and that is great, but at the same time we must understand that we are inevitably opening up ourselves for the rest of the world. This can create problems that did not exist before [3]. One problem is that the world is opening up for everyone – for people who are irresponsible, careless or even possessed of criminal intent. Computer systems and the information stored therein can become a weapon of the mind, so to speak, as well as an object of attack. Let us consider just a few of the many types of "computer crime" [4]:

- Unsanctioned access to information stored in a computer;
- Placement of "logical bombs" into programs that are set to "explode" under pre-determined circumstances and that can fully or partly knock out entire computer systems;
- Development and dissemination of computer viruses;
- Criminal negligence in developing, manufacturing and using computer systems that leads to serious consequences;
- Forgery of computer information;
- Theft of computer information.

Even if Latvia had laws against these types of activity (and, in fact, there is not a single paragraph of

Latvian law to make them illegal), it would not be easy to locate all of the violations and to prove their existence. The idea that information is an object with certain rights attached to it is new and unusual in global law. Information as such is not an object of ownership rights (except in the case of copyright and patent rights). If this is true, then there can be no theft of information, can there? Has something been stolen if in a subway car one person reads a page of a book over another person's shoulder? The book and all of its letters have remained in the hands of the owner of the book.

## THE PRINCIPLES OF OPENNESS AND PRIVACY (THE SWEDISH EXPERIENCE)

The openness principle [5,6], simply put, is that everyone has the right to access public (i.e., government and local government) documents. The only exception is documents which are declared to be secret or confidential in order to protect national security or sensitive private data about individuals. People who demand access to public documents should not, with very few exceptions, be made to identify themselves or to explain why they want to access the respective documents. Government officials who hand out documents should have no right to make inquiries in those directions. The openness principle also states that information must always be available in the form of a physical document. The law sets out the types of information that are confidential, but clerks have no right to classify documents (i.e., make them confidential) on their own.

With respect to privacy rights, on the other hand [5,6], Sweden has held censuses, generally speaking, every five years since the mid-19th century. As part of this process, the institution that organizes the censuses, Statistiska Centralbyran, passes out questionnaires. Residents are mandated by law to fill out these documents. During the 1970 census, there were loud protests about some of the questions that were on the questionnaire, as well as about the intended use of the information that was collected. In the end, a Data Act was adopted in 1973 that regulates the creation and maintenance of personal registers in computers. The basic idea in the Data Act is the inviolability of personal privacy rights. This is part of a legal doctrine that speaks to the legal sanctity of a person"s private life.

Sweden has no overall law on the defense of privacy rights, but a whole range of individual laws has been adopted, including the aforementioned data law. Supervision of the Data Act law is the job of a Data Inspection that was established in 1973.

The Data Act applies to work with automated data that involve individual persons, as well as the collection of data for such registers. If any activities in this field might touch upon a person"s private life in an unacceptable way, the Data Inspection issues normative documents to regulate the process. The law states that no activities may take place that offend or demean an individual. The data law also states that personal registers may be established only if there is no reason to believe that there will be any unacceptable impact on the private lives of individuals.

All institutions wishing to establish registers must first receive a license from the Data Inspection. Some registers are held to be particularly likely to have unacceptable impact on people's private lives. In order to establish such registers, institutions must receive not only the license, but also special authorization from the Data Inspection. The merger of various personal registers also requires authorization, except if the merger is ordered by law or by a normative document of the Data Inspection, or if the merger is approved by the parties who are registered in the register. Registers that are established on the basis of parliamentary or governmental decisions do not require the authorization of the Data Inspection, but it does have the right to pass judgment on such registers and, in some instances, to issue regulatory documents with respect to their use.

The Data Act sets out only the basic principles of this issue, and it contains very few rules concerning the issuance or denial of authorization. This gives the Data Inspection considerable freedom in interpreting and applying the law. It might also be added that the Data Inspection has only one full-time clerk-attorney. All of the other employees work on contract and are IT experts.

## LATVIAN LAW: THE EXISTING AND THE NECESSARY

It could not be said that there are no laws at all to regulate IT matters in Latvia. The following laws are more or less closely associated with this issue:

- The law "On author rights and neighboring rights"

The law [7] speaks to the right of individuals to use computer programs and, in this respect, is very important in the field of IT. The law states that programs must be purchased, not copied. The proof that a program has been obtained lawfully is a license issued by the author of the program or by successor of the author rights. Each program on every computer must be licensed, except in the case of so-called site or volume licenses. The law sharply diverges from the practice of other countries in the world, as well as the European Communities, concerning the issue of programs that have been created by salaried employees.

The law also sets out a procedure for the collection of damages, including lost profits, in the case of copyright violations. Courts are authorized to confiscate and destroy infringing computer programs and to order that materials used in connection with the programs (meaning the computers) can be turned over to the complainant in satisfaction of his demand for damages. The equipment can also be turned over to charity organizations or the state.

- The law "On patents"

The law protects the rights of inventors. This law has little effect in Latvia, because there will be no electronic factories here for a long time to come, and computer programs are

not protected in the law [8]. In this respect the law differs from legislation in the United States.

• The law "On telecommunications"

The law [9] is of secondary significance in the area of computer programs. The main point of the law is to set out the rights of telecommunications organizations.

• The Latvian criminal code

There are several important statements in the criminal code [10]. Statement 136 sets out sanctions for the violation of author rights, neighboring rights and patent rights (sentences start with fines of a sum that is equal to 20 times the government-approved minimum monthly wage in Latvia and end with imprisonment of up to six years with property confiscation). Statement 132 sets out sanctions (fines equal to five times the minimum monthly wage, ranging up to three years of imprisonment) for violations of the confidentiality of personal correspondence, information that is broadcast over telecommunications networks, as well as information or programs that are intended for electronic data processing.

• The law "On the Business Register of the Republic of Latvia"

Peculiar though this may seem, this law [11] does not contain a single word about the idea that the business register should be based on computer systems. The law permits the revelation to third parties of the names and addresses of members of company boards, which is a dodgy practice in terms of privacy rights.

• The law "On the Population Register"

The law [12] states that the register is an automated system for the registration of residents. There is nothing about the rules for data processing in the register.

Along with certain shortcomings in the aforementioned laws, there is also a complete absence of several types of law:

• A law on the openness of information;

• A law on the defense of personal data;

• A law (or other normative document) on applying juridical authority to electronic documents;

• A law (or a supplement to the criminal code) concerning unsanctioned access to computerized information (the current text of the criminal code states that the punishment for unauthorized break-in into a computer system, provided that it has not been done for purposes of greed and that it has not been the work of a government official, is a laughable fine of a sum equal to five minimum monthly wages).

Along with these laws, the country must also develop juridical regulations for the establishment and maintenance of personal registers. This will be difficult without the establishment of an organization similar to the Swedish Data Inspection.

## STANDARDS AS COMPROMISE

Why do we need standards, if their observance is not mandatory? Information technology, especially in the area of programming, provides an excellent answer to this question.

Academically educated computer programmers know (on the basis of theories which prove this) that there is no possible way to find out whether a computer program does what it is supposed to do and does not do anything that it is not supposed to do. We can run a program, first with one set of data, then with another and then with another, and each time we can see whether the program does what we expect of it. But there are infinite variations on the data that are input into the system, and we cannot know what the program will do when we enter the thousandth or millionth set of data. We can test a program until the second coming, but at some point every program must be released for use. Program users, naturally, will always demand that programs are better-made, checked and freed of all errors. But will they be prepared to await the end of the world and to pay for this theoretically endless work? From a juridical perspective, programmers simply cannot be held liable for errors in their programs for these very reasons. Of course, no country can permit fully irresponsible activities by its programmers, because losses caused by computer error can reach stratospheric heights. What to do?

If a user who has suffered losses sues the programmer, what should the judge (who usually is not and has no need to be a programming specialist) do? The judge will ask experts whether the programmer had any guaranteed way to avoid the error and the resulting losses. The experts will honestly reply that theoretically speaking, there is no such way. The judge will then ask whether the programmer has done everything that was sensibly necessary to avoid the possibility of losses. What does "sensibly necessary" mean? That is where standards come in. Standards define the needs of the current level of technological development, the compromise between what is desired and what is possible, the agreement between the programmer and the user – all of this based on the cumulative experience of the world. If programmers are disciplined in fulfilling the demands that are set out in programming standards, judges will have to find them blameless and to conclude that the losses were caused accidentally. If the programmer has failed to observe the standards that are addressed in the respective job agreement, however, then there will be reason to find him guilty of negligence.

## NATIONAL SYSTEMS OF IT STANDARDS

Should Latvia have its own IT standards? It would probably be more precise to ask whether Latvia even has sufficient resources to develop such standards. Indeed, Latvia's hopes of joining the European Communities suggest that the efforts should be directed toward harmonizing Latvian practices with international standards.

This means that only those standards which are needed for Latvia as a country of Latvians should be developed. Clearly, such standards will almost exclusively involve the standardized use and application of the Latvian language in computer systems. Other standards will certainly be translations of the standards of the ISO or

the European Communities or, if standards prove to be applicable more seldom or more narrowly, they will be given national status even without translation.

It is important to note that international standards are nothing more than recommendations in the countries in which they are applied, but standards which are given national status are in full effect. As we said, however, even such standards do not have the force of law.

The standardization process in Latvia is governed by the Latvian National Center of Standardization and Metrology, which exists under the auspices of the Ministry of Economics. This institution represents Latvia at the ISO (the international standardization body) and the CEN (the analogous European institution), maintains and disseminates the texts of standards, and organizes the development of national standards and the adaptation of international standards. In each sector of the economy where standardization is an issue, technical committees are usually established. In the IT sector, there is an Information Technology Standardization Technical Committee that was established in May 1995. For the purposes of clarification, the committee states in its bylaws that its operations correspond exactly to those of the ISO/IEC Joint Technical Committee One. The sector governed by that body, as is known, is referred to as "information technology".

The activities of the Latvian organization, however, tend to be focused on a few directions, each of which is handled by a sub-committee. The first subcommittee (SC1) deals with the development of standards to govern Latvian IT terminology. For understandable reasons, this work actively involves the Terminology Commission of the Latvian Academy of Sciences and the Informatics Sub-Committee of same. An electronic data base of terms is maintained, disseminated and expanded, and terminology dictionaries are published [14]. Latvia still does not have an overall standard of terminology, however.

The second sub-committee (SC2) deals with the use of the Latvian language in computer work. The sub-committee"s specialists are currently reviewing and updating standards which they themselves helped to develop prior to the creation of the standardization committee [15-19].

The seventh sub-committee (SC7) works to develop standards for software engineering. The committee has decided to base Latvian standards on those of the ISO and the IEEE [20-23]. A series of standards important in the area of programming has been approved in March 1996 [24-34].

It should be emphasized once again, however, that these and other activities in standardization and terminology do not create mandatory laws. Only the government can declare standards to be mandatory, as it has, indeed, done, through Cabinet of Ministers regulations [35].❑

## REFERENCES


1. 91/250/EEC Council Directive on the legal protection of computer programs, 14 May 1991.

2. Fleischmann, A. "Personal data security: divergent standards in the European Union and the United States", in *Fordham International Law Journal*, Vol. 19, No. 1, October 1995, pp. 143-180.

3. Brunnstein, K. and Sint, P.P. (eds.) Proceedings of the KnowRight '95 Conference "Intellectual Property Rights and New Technologies". Oldenburg, Vienna and Munich: Oesterreichische Computer Gesellschaft (1995).

4. Baturin, J.M and Zhodzishskij, A.M. Kompjuternaja prestupnostj i kompjuternaja bezopasnostj. Moscow: Juridicheskaja literatura (1991), in Russian.

5. Borzovs, J. "Datorizēti personreģistri un personas neaizskaramība Zviedrijā" (Computerized personal registers and privacy in Sweden), in *Atklājums*, No. 4, 1992, pp. 104-106, in Latvian.

6. Persson, G. Computerised Personal Registers and the Protection of Privacy. Preprint No. 344, Svenska Institutet (1991).

7. The law of the Republic of Latvia "On Author Rights and Neighboring Rights".

8. The Patents law of the Republic of Latvia.

9. The law of the Republic of Latvia "On Telecommunications".

10. The Latvian criminal code.

11. The law of the Republic of Latvia "On the Business Register of the Republic of Latvia".

12. The law of the Republic of Latvia "On the Population Register".

14. Baumgarts, V., et. al. Angļu-krievu-latviešu skaidrojošā vārdnīca "Datu pārraides un apstrādes sistēmas" (English-Russian-Latvian dictionary on data transmission and processing systems). Riga: A/s SWH Informatīvās Sistēmas (1995).

15. Latviešu valoda un standarti (The Latvian language and standards). Informatics Department of the State Chancellery of the Republic of Latvia in conjunction with the Tilde Company (1995), in Latvian.

16. LVS 8-92 8 bit coded graphic character set for Baltic Sea region countries.

17. LVS 23-93 Latvian keyboard for computers.

18. LVS 24-93 Latvian language support for computers.

19. RST 1040-90 8 bit code tables for information display in computers (in Russian)

20. ISO/IEC 12207:1995 (E) Information Technology - Software life cycle processes.

21. IEEE 1498/EIA IS 640:1995 Trial Use Standard - Standard for Information Technology - Software Life Cycle Processes - Software Development - Acquirer-Supplier Agreement.

22. ISO/IEC 12119:1994(E) Information technology - Software Packages - Quality requirements and testing.

23. IEEE Standards Collection "Software Engineering" (35 pieces).

24. LVS 65:1996 Software Quality Assurance Plan.

25. LVS 66:1996 Software User Documentation.

26. LVS 67:1996 Software Project Management Plan.

27. LVS 68:1996 Software Requirements Specification Guide.

28. LVS 69:1996 Software Configuration Management Guide.

29. LVS 70:1996 Software Test Documentation.

30. LVS 71:1996 Software Verification and Validation Plan.

31. LVS 72:1996 Recommended Practice in Software Design Description.

32. LVS 73:1996 Software Unit Testing.

33. LVS 74:1996 Software Reviews and Audits.

34. LVS 75:1996 Operational System Description.

35. Regulation No. 70 of the Cabinet of Ministers of the Republic of Latvia (adopted on 19 March 1996), Regulations concerning the order for assigning the status of computerized information system of state importance and the requirements for technical implementation.

**Starpdisciplinārie pētījumi: Datorzinātnes**

LATVIJAS ZINĀTŅU AKADĒMIJAS VĒSTIS. A. — 1997, 51. sēj., 3./4. (590./591.) nr.. 7.—10. lpp.

# DATORA IZMANTOŠANAS PRAKSE LZA TERMINOLOĢIJAS KOMISIJAS INFORMĀTIKAS APAKŠKOMISIJAS DARBĀ

Rudīte Čevere, Juris Borzovs, Marija Lučkina

Terminu izstrādāšana dažādām nozarēm, līdzīgi kā vairums valodniecības uzdevumu, ir radošs darbs, kas nav pilnībā formalizējams un nododams datoru ziņā. LZA Terminoloģijas komisijas (TK) Informātikas apakškomisija savā darbībā cenšas vadīties pēc terminoloģijas izstrādes vispārējiem principiem, kuri aprakstīti [1], kā arī ievērot konkrētās nozares īpatnības, kas minētas rakstā [2].

Vienlaikus tās situācijas, kuras rodas TK darba laikā, ir saistītas ar tipiskiem lielu informācijas apjomu glabāšanas, apzināšanas, meklēšanas un salīdzināšanas uzdevumiem, tādējādi tas ir darbs, kura veikšanā ne tikai ir iespējama, bet pat nepieciešama datoru izmantošana. Pat samērā šauras nozares terminu izstrādāšanā nākas darboties ar tik lielu informācijas apjomu, ka cilvēkam ar saviem spēkiem vien to ir grūti veikt pietiekami operatīvi.

Kā raksturīgus piemērus var minēt šādas terminu izstrādāšanas gaitā bieži sastopamas situācijas:

1) terminam ir izvēlēts par tulkojumu latviešu valodas vārds, kurš būtu precīzāks un brīžam pat vienīgais atbilstošais tulkojums citam radnieciskam angļu valodas terminam (piemēram, termini *mark, label, tag* u.tml.). Šī situācija var gadīties tad, ja kādas radnieciskas terminu grupas apspriešana notiek, izskatot šos terminus dažādās komisijas sēdēs;

2) terminam tā apspriešanas laikā tiek piedāvāts kāds variants, tas argumentēti tiek noraidīts un apstiprināts cits tulkojums, taču pēc kāda laika tiek ierosināts jau apstiprinātajam terminam mainīt tulkojumu uz iepriekš piedāvāto. Dabiski, ka komisijas locekļi ne vienmēr var atcerēties, ka šāds piedāvājums jau ir bijis un ar kādiem argumentiem tas ticis noraidīts;

3) termina apspriešanas laikā ir grūti novērtēt, kādās terminoloģiskās vārdkopās vēl šis vārds varētu būt lietojams un cik labskanīgs būs šis lietojums.

Līdzīgu problēmu uzskaiti varētu turpināt, tomēr galvenais ir mēģināt atrast palīglīdzekļus, kā atvieglot un veicināt terminu izstrādāšanas darbu. Šādu palīglīdzekļu galvenais uzdevums ir krāt pēc iespējas plašāku informāciju par terminu izstrādes gaitu un jau apstiprinātajiem terminiem, veikt šīs informācijas apstrādi un sagatavot dažāda veida apkopojumus un citus uzziņu materiālus, kā arī nodrošināt ērtu pieeju tādai palīginformācijai kā dažādu valodu vārdu krājumi, tulkojošās un skaidrojošās vārdnīcas u.tml. Speciāli sagatavota palīginformācija Terminoloģijas komisijai ir nepieciešama arī tādēļ, lai savā praktiskajā darbībā spētu ievērot tādas vadlīnijas kā, piemēram, šīs trīs:

1) atšķirīgiem terminiem izcelsmes valodā jācenšas veidot atšķirīgus terminus latviešu valodā;

2) jaundarināmais termins nedrīkst tikt piedāvāts atrauti no apkārtnes, no tam tuviem un analoģiskiem terminiem, kā arī no jau lietojamiem terminiem, jāpārliecinās, vai vārds "labi strādā", t.i., vai ērti lietojams vārdu savienojumos, lokāms, vai no tā saknes veidojams lietvārds, darbības vārds, īpašības vārds u.tml.;

3) praksē jau lietojamos terminus vēlams nemainīt, ja tam nav būtiska pamatojuma.

Tradicionāla pieeja šādu palīglīdzekļu izstrādē ir terminoloģijas vārdnīcas veidošana datorizētas datubāzes formā. Veidojot šādu datubāzi, vispirms tiek formulēti galvenie uzdevumi, kas ar tās palīdzību tiks veikti. Kā galvenos var minēt šādus uzdevumus:

- apstiprināto terminu un to skaidrojumu bāzes uzturēšana;
- apstiprināšanas vēstures uzturēšana;
- jaunveidojamo terminu saskaņošana ar jau esošiem terminiem;
- citu nozaru terminu respektēšana nozaru saskarapgabalos;
- iespējami plaša informācijas sagatavošana katrai TK sēdei;
- latviešu terminoloģijas kopbāzes izveidošana un uzturēšana;
- vārdnīcu sagatavošana izdošanai.

Informātikas apakškomisija jau vairākus gadus izmanto personālo datoru un īpaši izstrādātas datorprogrammas minēto jautājumu risināšanai. Izstrādājot dažādu nozaru datorizētas terminoloģiskas vārdnīcas, svarīgi ir ievērot virkni kopīgu prasību, kas nodrošina vārdnīcu vispusīgu izmantošanu, kā arī iespēju vēlāk apkopot terminoloģisko materiālu vienotā latviešu valodas terminoloģijas bāzē, kas ir viena no valodas fonda sa-

Rīgas Informācijas tehnoloģijas institūts

stāvdaļām. No vārdnīcu lietošanas viedokļa viena no galvenajām prasībām ir nodrošināt informāciju, kura nepieciešama tipveida darba seansu īstenošanai. Savukārt pēc terminoloģijas apakškomisijas darba pieredzes var nodalīt šādus tipveida darba seansus:

- vārdnīcas veidošana un papildināšana,
- terminu izstrādāšana,
- terminu izgūšana,
- terminu analizēšana,
- terminoloģijas vārdnīcu apakškopu formēšana, formatēšana un izdošana.

Galvenās informācijas vienības, kuras jāuztur terminu bāzē par katru bāzes elementu, t.i., terminu vai terminoloģisko vārdkopu, ir aprakstītas tabulā.

Vārdnīcas veidošana un papildināšana ir darba seanss, kurš drīkst būt pieejams ļoti šauram lietotāju lokam — tikai darbiniekam, kurš ir atbildīgs par vārdnīcas uzturēšanu (t.s. vārdnīcas administratoram). Lai nodrošinātu, ka ieraksti datubāzē tiks pievienoti vai mainīti tikai pēc nopietnas vispusīgas pārbaudes, šis darba režīms nedrīkst būt pieejams terminu apspriešanas laikā. Vārdnīcas veidošanai un papildināšanai kā ievades informācija ir jāsagatavo minētajā tabulā aprakstītā katra termina pamatinformācija un papildinformācija. Ievades laikā datorizētās vārdnīcas pārvaldības programma organizē nepieciešamo pārbaudi, piemēram, vai šāds termins vai terminoloģiskā vārdkopa jau atrodas vārdnīcā oriģinālvalodā (angļu vai citā valodā, kura ir izvēlēta kā dotās nozares terminu izcelsmes valoda); vai šī termina vai terminoloģiskās vārdkopas piedāvātais tulkojums jau ir izmantots vārdnīcā kā cita termina tulkojums u.tml. Ja atrod kādu no iepriekš formulētām situācijām, lietotājam izvada informāciju, kurā ir pateikta arī iespējamā tālākā darbība. Ja vārdnīcas administrators atzīst, ka sagatavotā ievades informācija nav pretrunā ar vārdnīcas saturu, tiek veikta termina ievietošana vārdnīcā tam atvēlētajā vietā, kā arī, ja iespējams, termina atzīmēšana visos esošajos skaidrojumos un visu jaunā termina skaidrojuma tekstā esošo potenciālo terminu pārbaude un ietveršana šķērsatsaucēs. Nepieciešamības gadījumā vārdnīcas administratoram ir tiesības atlikt termina ievadīšanu un atdot informāciju terminoloģijas apakškomisijai neskaidro jautājumu izlemšanai. Līdzīgi arī terminu dzēšanas gadījumā tiek veikta pārbaude, vai šāds termins tiešām ir atrodams. Pirms dzēšanas izpildes vēlreiz obligāti pārprasa apstiprinājumu. Sarežģītāka scenārija gadījumā var meklēt arī visus saistītos terminus — vārdkopas, kurās dzēšamais termins sastopams, un piedāvāt to apstrādi.

| Nr. | Informācijas vienība | Informācijas apraksts |
|---|---|---|
| | | *Pamatinformācija* |
| 1. | Termins latviski | Vārds vai vārdkopa latviešu valodā |
| 2. | Termins angliski | Vārds vai vārdkopa angļu valodā |
| 3. | Termins citā svešvalodā | Vārds vai vārdkopa valodā, kura tiek uzturēta konkrētās nozares terminoloģiskajā bāzē |
| 4. | ... | ... |
| 5. | Termina numurs (kods) | Numurs (kods), ko piešķir katram terminam — vārdam vai vārdkopai (unikāls šīs bāzes ietvaros). Terminus numurē pēc to sakārtojuma valodā, kura šīs nozares terminiem ir izvēlēta par pamatvalodu, t.i., terminu rašanās valodu. Piemēram, informātikas nozares terminiem tā ir angļu valoda |
| 6. | Stāvoklis | Pazīme, kas norāda, kādā apspriešanas stadijā termins atrodas (piemēram, izvirzīts apspriešanai, tiek apspriests, apstiprināts apakškomisijā, apstiprināts komisijā utt.) |
| 7. | Apstiprināšanas datums | Datums, kurā termins ir apstiprināts apakškomisijā, apstiprināts komisijā vai arī veikta kāda cita nozīmīga darbība ar šo terminu |
| | | *Papildinformācija* |
| 1. | Skaidrojums | Termina skaidrojums (vēlams latviešu valodā). Pēc savas nozīmes skaidrojums pieder pamatinformācijai, papildinformācijā ievietots tādēļ, ka tā ir apjomīgākā informācijas daļa, kas prasa lielus resursus un darbietilpību tās ievadei, tādēļ var būt grūti realizējama |
| 2. | Termina avots | Norāda atsauci uz literatūru vai citu avotu, no kura ņemts pamattermins un/vai tā skaidrojums |
| 3. | Termina vēsture | Informācija par termina iepriekšējo apspriešanas gaitu |
| 3.1. | Datums | Iepriekšējais termina apstiprināšanas datums |
| 3.2 | Iepriekš apstiprinātais termins | Vārds vai vārdkopa latviešu valodā |
| 3.3 | Piezīmes | Termina izvēles vai noraidījuma motivācija vai kāda cita būtiska informācija |

Terminu izstrādāšanas atbalsts ir darba režīms, kura laikā nav iespējams izdarīt nekādas izmaiņas datubāzē. Tas ir domāts terminu apspriešanai, lai sagatavotu informāciju pirms Terminoloģijas komisijas sēdes, kā arī operatīvai informācijas meklēšanai tieši komisijas sēdes laikā.

Jauna termina vai arī jau apstiprināta termina atkārtotai apspriešanai sagatavojamās informācijas sastāvā ietilpst:

1) jaunais vai maināmais pamattermins ar visiem tulkojumiem un skaidrojumu;

2) visas terminoloģiskās vārdkopas, kurās ietilpst šis termins vai atsevišķi tā vārdi (vēlams ar skaidrojumiem);

3) visi to terminu skaidrojumi, kuros izmantots apspriežamais termins;

4) visu angļisko (vai citas oriģinālvalodas) terminu — sinonīmu — grupa no apstiprinātās terminoloģiskās vārdnīcas ar visiem tulkojumiem un, vēlams, skaidrojumiem (piemēram, *object, entity, item, unit* utt.);

5) visi piedāvātie latviskie šī termina tulkojumi — sinonīmi — un to izmantošana jau apstiprinātajā vārdnīcā;

6) termina izstrādāšanas vēsture (ja tā tiek uzturēta).

Terminu izguve ir visplašākajam lietotāju lokam domātais darba režīms, kura laikā līdzīgi kā iepriekš nav iespējams izdarīt izmaiņas vārdnīcas saturā.

Tipiskie izgūstamās informācijas varianti ir šādi:

1) uzdota termina vai terminoloģiskās vārdkopas tulkojums vienā noteiktā valodā vai visās valodās;

2) noteikta terminu apakškopa, piemēram, visas terminoloģiskās vārdkopas, kurās ietilpst kāds uzdots termins;

3) uzdota termina vai terminoloģiskās vārdkopas skaidrojums;

4) noteikta terminoloģiskā ligzda.

Terminu analizēšana ir darba režīms, kura lietotāji galvenokārt ir valodnieki. Ja datorizētā vārdnīcā par katru vārdnīcas elementu tiek uzturēta tabulā aprakstītā informācija, izstrādājot samērā vienkāršas pro- grammas, var nodrošināt palīdzību dažādu nozaru terminoloģisko vārdnīcu veidošanā, nozaru kopējo terminu izdalīšanā, analīzē un saskaņošanā u.tml.

Var veikt arī terminoloģiskā materiāla analizēšanu no tīri valodnieciska viedokļa — iegūt izmantotās vārdu darināšanas statistiku, lietošanas biežumu utt.

Terminoloģijas vārdnīcu apakškopu formēšana, formatēšana un izdošana ir vēl viens datorizētās terminoloģijas vārdnīcas lietošanas režīms, kuram ir liela praktiska nozīme. Ar programmu palīdzību ir iespējams ērti atlasīt noteiktu terminu apakškopu, veikt to sakārtošanu, veidot indeksus un šķērsatsauces, kā arī iegūt daudzveidīgi formatētu tekstu, izmantojot dažādus rindkopas stilus, fontus, burtu augstumus, kā arī iespējas lietot pustrekno druku, kursīvu utt.

Rīgas Informācijas tehnoloģijas institūtā izstrādātā datorizētā vārdnīca ir realizēta datubāzes FoxPro vidē un ir uzskatāma par šādas vārdnīcas vienkāršotu prototipu, tomēr jau pašlaik tās lietošana LZA TK Informātikas apakškomisijas darbā ir atzīstama par ļoti noderīgu.

Kā dažus no galvenajiem rezultātiem var minēt šādus:

Nesen izdotā "Angļu-krievu-latviešu skaidrojošā vārdnīca: Datu pārraides un apstrādes sistēmas" tika gatavota iespiešanai, izmantojot tikai personālā datora iespējas. Īpašas programmas palīdzēja veikt rūpīgas un visaptverošas šķērspārbaudes, kas ļāva novērst lielu daudzumu neuzmanības kļūdu, kā arī viegli sagatavot latviešu un krievu terminu rādītājus (vārdnīca sakārtota angļu terminu secībā).

Visa pašreizējā informācijas tehnoloģijas apstiprināto terminu kopa glabājas vienā noteiktā datorā, un to uzturēt (papildināt, labot u.tml.) drīkst tikai viens pilnvarots speciālists, toties šīs terminu kopas kopijas tiek brīvi izplatītas datortīklos (tās iespējams saņemt arī uz disketēm SWH IS Komerccentra salonā un pie pārstāvjiem rajonu pilsētās). Domstarpību gadījumos par etalonu tiek atzīta jau minētajā datorā glabātā terminu kopa, kurā izmaiņas drīkst izdarīt — pat acīm redzamu kļūdu gadījumā — tikai ar apakškomisijas dokumentētu lēmumu. Pēkšņi labojumi terminu apspriešanas gaitā ne tikai nav pieļaujami, bet ir tehniski neiespējami. Tas ļoti disciplinē darbu un nodrošina esošās terminu bāzes integritāti.

Apspriežot kārtējo jauno terminu, tiek izmantots dators, kurā atrodas svaigākā etalona kopija. Izmantojot jau minētās programmas, tiek veikta apspriežamā termina pārbaude, kas pat visai vāja datora gadījumā neaizņem vairāk par dažiem desmitiem sekunžu (terminu bāzē pašlaik ir pāri par 3000 terminu), pie tam šo pārbaudi var veikt, nepārtraucot apspriešanu. Tā tiek panākts, ka dažādiem angļu terminiem netiek piekārtots viens un tas pats latviešu termins, ka viens un tas pats latviešu vārds netiek izmantots atšķirīgu jēdzienu apzīmēšanai u.tml.

Lai terminu projektu iepriekšējā apspriešanā iesaistītu iespējami plašu interesentu loku, jo ne visi atrod laiku piedalīties apakškomisijas sēdēs, kaut tās visas ir atklātas un izteikties un balsot atļauts ikvienam klātesošam, elektroniskā formā sagatavotie priekšlikumi tiek darīti pieejami caur t.s. Interneta globālo tīmekli (WWW — *World Wide Web*). Ikviens interesents var savus priekšlikumus darīt zināmus apakškomisijai — personiski, telefoniski vai izmantojot e-pastu.

Lai nodrošinātu latviešu terminu kopbāzes izveidošanu, jau šodien katras nozares terminu izstrādātāji var un viņiem ir nepieciešams ievērot vismaz šādus nosacījumus:

1) katras nozares terminu bāze jāveido datorā;

2) jāvienojas par informāciju, kura tiek uzglabāta par katru terminu, turklāt programmatūras vide katrai nozares terminu bāzei sliktākā gadījumā var būt dažāda, bet jāuzkrāj viena un tā paša veida informācija;

3) jāizstrādā kopīgi pamatprincipi globālās terminu bāzes izstrādei, piemēram, ar Interneta līdzekļiem (WWW);

4) jāizstrādā katras nozares bāzes saskarne ar kopējo bāzi.

Rīgas Informācijas tehnoloģijas institūts būtu ar mieru līdztekus informācijas tehnoloģijas terminiem uzturēt arī visu latviešu terminu kopbāzi, kā arī sniegt nepieciešamās konsultācijas un dot iespēju arī citu nozaru apakškomisijām lietot vārdnīcas prototipa programmatūru.

VĒRES


1. Skujiņa V. *Latviešu terminoloģijas izstrādes principi.* Rīga: Zinātne, 1993.
2. Borzovs J. Lai iegūst latviešu valoda! *Datortehnika.* 1993. 3: 7, 8.
3. Čevere R., Borzovs J. Lietotāja pamatprasības tipveida darba seansiem ar datorizētu terminoloģisko vārdnīcu. Grām.: *Baltistica VII: VII Starptautiskais baltistu kongress.* Rīga, 1995. 25.–27. lpp.
4. Baumgarts V., Baums A., Gobzemis A., Fricnovičs G., Ilziņa L. (sast.) *Angļu-krievu-latviešu skaidrojošā vārdnīca: Datu pārraides un apstrādes sistēmas* / Terminol. konsult. V. Skujiņa. Rīga: A/s SWH Informatīvās sistēmas, 1995.


Iesniegts 18.02.97

## PRACTICAL USAGE OF COMPUTERS IN THE WORK OF THE INFORMATICS SUBCOMMITTEE OF THE TERMINOLOGY COMMITTEE OF THE LATVIAN ACADEMY OF SCIENCES

Rudīte Čevere, Juris Borzovs, and Marija Lučkina


Summary

While developing Latvian terminology in various areas, members of the Terminology Committee of the Latvian Academy of Sciences must analyze a great amount of information. This article looks at the development of databases as computer-aided tools for this purpose. The general requirements for such dictionaries, as well as the main information units required for each term, are described. Practical experience gained through use of the dictionary in the work of the Informatics Subcommittee of the Terminology Committee is also discussed.


Rīga Information Technology Institute

*Латвийский математический ежегодник, 1984, вып. 28 (Рига), с. 157—161*

# ДИСКРЕТНАЯ МАТЕМАТИКА И АЛГЕБРА

УДК 618.3.06

**Ю. В. Борзов, Г. Б. Уртанс**

## ФОРМУЛЫ ОБРАБОТКИ ФАЙЛОВ. II

Данная работа является логическим продолжением работы [2]. Исследуются возможности автоматической записи функции, которую вычисляет программа ЭВМ, на непроцедурном языке спецификаций. В [2] был описан простой язык программирования обработки последовательных файлов (базовый язык [1]) и введен язык спецификаций низкого уровня (язык формул обработки файлов). Было доказано, что существует алгоритм, который для любой программы на базовом языке строит конечную систему формул обработки файлов (ФОФ), которая задает в точности то же преобразование данных, что и сама программа. Здесь будет доказано, что при естественных ограничениях на системы ФОФ существует алгоритм, который для любой «естественной» системы ФОФ строит ту программу на базовом языке (или ей функционально эквивалентную), из которой система могла быть получена алгоритмом из [2]. Основные определения содержатся в [2] и ввиду их громоздкости в настоящей работе не повторяются.

### 1. НЕПРОТИВОРЕЧИВЫЕ ФОФ

Очевидно, что не всякая система ФОФ является функцией программы базового языка. Для иллюстрации рассмотрим следующую ФОФ: $\langle\cdot\langle\cdot U(1)\cdot\rangle\langle\cdot U(0)\cdot\rangle\cdot\rangle$.

Эта ФОФ задает функцию программы, которая независимо от входных данных выдает на выходной файл $U$ произвольную последовательность нулей и единиц, из-за недетерминированности такую программу на базовом языке написать невозможно.

Следовательно, для того чтобы из всех ФОФ выделить «хорошие», нужно наложить дополнительные требования на ФОФ.

Для определения непротиворечивости введем несколько вспомогательных понятий.

Определение 1. Дуальными будем называть следующие элементы или последовательности элементов ФОФ:

1) $A^*$ и $A_{k_A+1}$,

$A^*$ и $\bar{A}_j A_{k_A}$,

$A^*$ и $\bar{B}_j A_{k_A+1}$

где A, $B$ — имена входных файлов, $k_A$ — число доступных записей файла перед обозреваемым элементом, а $j \in N$;

2) $\lfloor A < B \rfloor$ и $\lfloor A \geqslant B \rfloor$, где A, $B$ — либо записи, либо константы.

Определение 2. Две формулы будем называть графически дуальными, если существует такой $i \in N$, что подформулы, содержащие начала формул до $i$-го элемента, совпадают графически; $i$-е элементы (или $i$-й и последовательность, содержащая $i$-й и $(i+1)$-й элементы) являются дуальными и $i$-е элементы ($i$-й и $(i+1)$-й элементы) не находятся в цикле.

Определение 3. Систему ФОФ будем называть непротиворечивой, если каждые две ФОФ из этой системы являются графически дуальными.

Разделим все циклы в ФОФ на три группы.

Определение 4. Цикл будем называть альтернативным, если существует $n$ циклов ($n>1$) $c_1, c_2, \ldots, c_n$, что этот цикл представляется в форме $\langle \cdot c_1 c_2 \ldots c_n \cdot \rangle$.

Циклы $c_1, c_2, \ldots, c_n$ будем называть составляющими.

Определение 5. Циклы, не являющиеся ни альтернативными, ни составляющими, будем называть простыми.

Определение 6. ФОФ будем называть непротиворечивой, если

1) тело каждого простого цикла графически дуально с подформулой, содержащей ФОФ начиная с $(j+1)$-го элемента (где $j$ — номер последнего элемента данного цикла);

2) для каждого альтернативного цикла тела всех составляющих циклов и подформула, содержащая ФОФ начиная с $(j+1)$-го элемента ($j$ — номер последнего элемента данного цикла), попарно графически дуальны.

Рассмотрим два примера:

1) $\langle \cdot \langle \cdot U(1) \cdot \rangle \langle \cdot U(0) \cdot \rangle \cdot \rangle$
противоречива, так как $u(1)$ и $u(0)$ не являются графически дуальными;

2) $\langle \cdot \langle \cdot A_1 \lfloor A_1 < 4 \rfloor U(1) \bar{A}_1 \cdot \rangle \langle \cdot A_1 \lfloor A_1 \geqslant 4 \rfloor U(0) \bar{A}_1 \cdot \rangle \cdot \rangle A^*$ непротиворечива, так как $A_1 \lfloor A_1 < 4 \rfloor U(1) \bar{A}_1$, $A_1 \lfloor A_1 \geqslant 4 \rfloor U(0) \bar{A}_1$, $A^*$ являются попарно графически дуальными.

Лемма 1. Если ФОФ получена из программы базового языка алгоритмом А [2], то она непротиворечива.

Для доказательства леммы рассмотрим работу алгоритма А на дереве реализуемости (Д(П)). Пусть при построении данной ФОФ мы остановились на вершине $q_k$ Д(П). Если эта вершина не явля-

ется обрывающей, то она не является началом цикла и поэтому не будем ее рассматривать. Если вершина $q_k$ есть обрывающая и ей соответствует одна оборванная вершина, то при работе алгоритма А будет построен простой цикл.

После построения цикла продолжение будет строиться снова начиная с вершины $q_k$, причем $j$ шагов мы будем двигаться по тому же пути, что и при построении цикла. Очевидно, что эти $j$ шагов тело цикла и соответствующая подформула будут совпадать. На $(j+1)$-м шаге мы свернем с ранее пройденного пути, следовательно, используем второй выход $(j+1)$-й команды в этой ветви.

Рассмотрим все команды с двумя выходами.

1) При обработке чтения с выходом «+» может быть написано

$$A_{k_A+1}, \quad \bar{B}_j A_{k_A+1} \quad \text{или} \quad \bar{A}_j A_{k_A},$$

при обработке чтения с выходом «—» — $A^*$. Ясно, что в этом случае формулы графически дуальны.

2) При обработке условия с обоими выходами тоже появляются дуальные элементы. Других команд с двумя выходами нет.

Отметим, что $(j+1)$-я команда не может находиться в цикле, который начался после обозреваемого. В начале обработки каждого цикла задается определенный путь от обрывающей до оборванной вершины и свернуть с него невозможно. Из этого следует невозможность появления дуальных элементов в таких циклах.

Из вышеописанного следует выполнение непротиворечивости для простых циклов.

Отметим также, что при альтернативном цикле такое же рассуждение можно провести для любой необходимой пары с учетом того, что альтернативному циклу в Д(П) соответствует обрывающая вершина с несколькими оборванными вершинами.

Лемма 2. Система ФОФ, полученная алгоритмом А, является непротиворечивой системой ФОФ.

Лемма доказывается аналогично лемме 1. Из обеих лемм следует, что А генерирует непротиворечивые системы непротиворечивых ФОФ.

## 2. РЕАЛИЗУЕМЫЕ ФОФ

Сначала определим состояния реализуемости в точках ФОФ (определение точки в ФОФ дано в [2]).

Определение 7. Определим для произвольной ФОФ $X$ путь $\alpha(X)$ следующим образом. Заменим элементы ФОФ на следующие команды базового языка:

1) $A_i$ — на $A \Rightarrow a_i$ с выходом «+»;

2) $\bar{A}_i$ — на $a_{i+1} \Rightarrow a_i$,
$a_{i+2} \Rightarrow a_{i+1}$,
$\vdots$
$a_{k_A} \Rightarrow a_{k_A-1}$;

3) $\lfloor A_i < B_j \rfloor$ ($\lfloor A_i < C \rfloor$) — на $a_i < b_j$ ($a_i < C$) с выходом «+»;
4) $\lfloor A_i \geqslant B_j \rfloor$ ($\lfloor A_i \geqslant C \rfloor$) на $a_i \not< b_j$ ($a_i \not< C$) с выходом «−»;
5) $\lfloor C < A_i \rfloor$ — на $C \Rightarrow c$ и $c < a_i$ с выходом «+»;
6) $\lfloor C \geqslant A_i \rfloor$ — на $C \Rightarrow c$ и $c \not< a_i$ с выходом «−»;
7) $U(A_i)$ ($U(C)$) — на $a_i \Rightarrow U(C \Rightarrow c$ и $c \Rightarrow U)$;
8) $A^*$ — на $A \Rightarrow a_{k_A+1}$ с выходом «−».

Построим состояния реализуемости для построенного пути (как в [1]). Состоянием реализуемости в $i$-й точке будем называть состояние реализуемости $j$-й команды пути $\alpha(X)$, если $j$-я команда есть последняя, полученная из $i$-го элемента.

Определение 8. ФОФ будем называть реализуемой, если соответствующий путь $\alpha(X)$ реализуем и для каждого цикла состояния реализуемости перед и после цикла совпадают.

## 3. ТЕОРЕМА О ПОСТРОЕНИИ ПРОГРАММЫ ПО ЕЕ ФУНКЦИИ

Теорема. Существует алгоритм $A_2$, который для каждой непротиворечивой системы непротиворечивых ФОФ $F$ строит программу $A_2(F)$, у которой $A(A_2(F)) = F$.

Опишем процедуру построения программы. Если в системе нет ни одной ФОФ, то программа зацикливается на всех входных данных.

Обрабатываем каждую формулу $F$ следующим образом, строя $A_2(F)$. Если очередным элементом ФОФ является

1) запись $A_i$, в данной точке ставим команду $A \Rightarrow a_i$ и рисуем вниз выход «+»;

2) условие конца файла $A^*$, ставим команду $A \Rightarrow a_{k_A+1}$ и рисуем выход вниз «−»;

3) гашение $\overline{A_i}$, ставим последовательность команд $a_{i+1} \Rightarrow a_i$, $a_{i+2} \Rightarrow a_{i+1}, \ldots, a_{k_A} \Rightarrow a_{k_A-1}$ и рисуем выход «+»;

4) условие $\lfloor X_i < Y_j \rfloor$, ставим команду $x_i < y_j$ и рисуем выход «+»;

5) условие $\lfloor X_i \geqslant Y_j \rfloor$, ставим команду $x_i < y_j$ и рисуем выход «−»;

6) результат $U(X_i)$ ($U(C)$), ставим команду $x_i \Rightarrow U(c \Rightarrow U)$;

7) если в ФОФ необработанных элементов не осталось, ставим команду СТОП.

Если при выполнении этих действий перед рассматриваемым элементом находится '⟨·', то выясняем, какой цикл начинается.

I. Если начинается простой цикл, то его обрабатываем согласно 1)—6), пока не дойдем до конца цикла. В конце цикла рисуем дугу вверх в первую команду, полученную в данном цикле.

II. Далее продолжаем обработку ФОФ, находясь на команде, соответствующей началу данного цикла (первой команде, полученной при обработке данного цикла).

III. Если это альтернативный цикл, то обрабатываем каждый составляющий цикл согласно I. По окончании переходим к II.

Если при обработке некоторого цикла начинается еще один цикл, то прерываем обработку этого цикла, запоминаем ситуацию и начинаем обработку следующего цикла. Когда данный цикл будет обработан, продолжаем обработку предыдущего.

В некоторых случаях необходимо будет двигаться по уже нарисованным выходам. Тогда совпадение поставленных команд и тех, которые в данный момент нужно поставить, гарантирует непротиворечивость ФОФ и системы ФОФ. На этом описание процедуры окончено.

Теперь будем строить ФОФ полученной программы. По алгоритму А, сначала следует построить Д(П). Легко можно убедиться, что структура Д(П) почти совпадает со структурой самой программы, только выходы, идущие вверх (полученные при обработке конца цикла), будут оборваны по повторению (это следует из того, что в начале и в конце каждого цикла в начальной ФОФ состояния совпадали).

Ни одна ветвь не будет оборвана как нереализуемая. Это следует из того, что в ФОФ во всех точках состояния реализуемы. Наконец, ни одна ветвь не будет оборвана по повторению раньше, чем в выходах, идущих вверх, потому что во всей ветви не повторяется одна и та же команда (с тем же номером).

При переходе из Д(П) в ФОФ команды в Д(П) снова перейдут в начальные элементы ФОФ (которые были в исходной ФОФ), появятся и те же самые циклы. Это следует из инверсии алгоритмов перехода в ФОФ и перехода в Д(П) (в чем можно убедиться из рассмотрения I—III обоих алгоритмов, а именно они задают структуру программы ФОФ).

Таким образом, $A_2(A(F))=A$ и $A_2$ действительно строит нужную программу. На этом доказательство теоремы закончено.

## СПИСОК ЛИТЕРАТУРЫ

1. Барздинь Я. М., Бичевский Я. Я., Калниньш А. А. Построение полной системы примеров для проверки корректности программ. — Учен. зап. ЛГУ им. П. Стучки, 1974, т. 210, с. 152—187.
2. Борзов Ю. В., Уртанс Г. Б. Формулы обработки файлов. — Латв. мат. ежегодник, вып. 27, 1983, с. 210—215.

*Вычислительный центр*
*ЛГУ им. П. Стучки*

*Поступила 23.09.83*

*Латвийский математический ежегодник, 1983, вып. 27 (Рига), с. 202—209*

УДК 618.3.06

**Ю. В. Борзов, Г. Б. Уртанс**

# ФОРМУЛЫ ОБРАБОТКИ ФАЙЛОВ

В последние годы значительное внимание было уделено разработке методов наглядного описания функции, задаваемой программой [3—8]. В случае обработки данных (КОБОЛ, ПЛ/1 и т. п.) входными значениями и результатами работы программы являются файлы. Следовательно, функция, реализуемая программой, представляет собой преобразование одних файлов в другие. Ниже рассматривается одна из возможных форм записи такого преобразования. В отличие от самого текста программы, тоже являющегося записью этого же преобразования, используются только понятия преобразуемых данных — файл, запись, поле и выражения над ними.

## 1. ЯЗЫК ПРОГРАММИРОВАНИЯ

Понятно, что описание функции, которую реализует программа (функции программы), существенно зависит от допустимых структур данных и операций над ними. Поэтому элементы описания, по-видимому, должны различаться для различных языков программирования. Мы сосредоточили внимание на так называемом базовом языке [1, 2], являющемся упрощенной моделью языков обработки последовательных файлов. Этот язык характерен тем, что для него разрешима проблема построения полной системы тестов.

Для полноты изложения приведем описание базового языка.

### 1.1. Базовый язык

Мы опишем некоторую абстрактную машину (типа машины Тьюринга), работающую с последовательными файлами. Эта машина называется базовой машиной, а язык программирования, порожденный ею, — базовым языком.

Определим базовую машину (рис. 1). Она состоит из управляющего блока, конечного числа односторонних входных лент $A, B, \ldots, C$ и конечного числа односторонних выходных лент $U, V, \ldots, Z$. Управляющий блок содержит конечное число ячеек $a, b, \ldots, v$, называемых внутренними ячейками. В каждую внутреннюю ячейку можно записывать произвольное целое число. Входные и выходные ленты состоят также из ячеек; в каждую ячейку ленты также можно записывать произвольное целое число. Ленты используются для изображения файлов. Будем говорить, что на ленте записано значение файла $(n_1, n_2, \ldots, n_r)$, если начиная с первой ячейки записаны числа $n_1, n_2, \ldots, n_r$, а остальные ячейки пустые (рис. 1).

Числа $n_1, n_2, \ldots, n_r$ будем называть записями файла. В дальнейшем, говоря о файле $F$, будем понимать файл, записанный на ленте $F$.

Определим теперь команды, которые может выполнять машина. Пусть $X$ — обозначение произвольной входной ленты; $Y$ — обозначение произвольной выходной ленты; $t, v$ — обозначения произвольных внутренних ячеек; $C$ — произвольное целое число (константа).

1) $X \Rightarrow t$ — содержимое обозреваемой ячейки ленты переписывается во внутреннюю ячейку $t$ (т. е. ячейке $t$ присваивается очередная запись файла $X$), и головка передвигается вправо. Команда имеет два выхода: выход «+», когда обозреваемая ячейка содержит целое число, и выход «−», когда обозреваемая ячейка пустая. В последнем случае значение ячейки $t$ не меняется.

2) $t \Rightarrow Y$ — содержимое внутренней ячейки $t$ переписывается в обозреваемую ячейку выходной ленты $Y$, а головка передвигается на одну ячейку вправо. Команда имеет один выход, который для определенности обозначим через «+».

3) $t \Rightarrow v$ $(C \Rightarrow v)$ — содержимое ячейки $t$ (или константа $C$) переписывается в ячейку $v$. Команда имеет один выход — «+».

4) $t < v$ $(t < C)$ — команда имеет два выхода: если содержимое ячейки $t$ меньше содержимого ячейки $v$ (или константы $C$), то управление передается по выходу «+», в противном случае — по выходу «−».

5) СТОП — машина останавливается. Команда не имеет выходов.

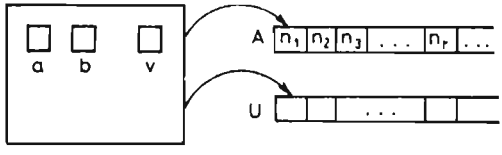



*Рис. 1.*

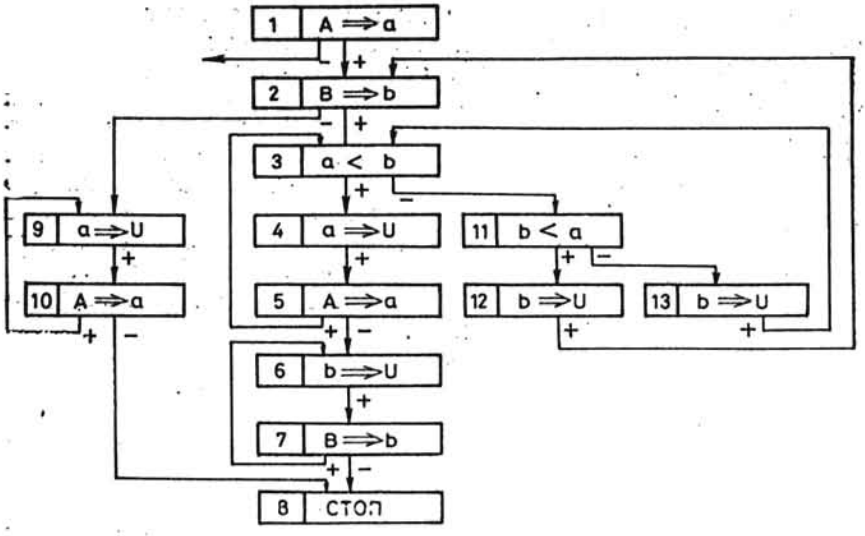



*Рис. 2.*

Программу можно наглядно представить граф-схемой. В качестве примера рассмотрим программу слияния двух отсортированных файлов из [1] (рис. 2).

## 2. ФОРМУЛА ОБРАБОТКИ ФАЙЛОВ (ФОФ)

Для определения формулы обработки файлов нам понадобится несколько вспомогательных определений.

Зафиксируем множество имен входных и выходных файлов.

Определим запись:

⟨запись⟩ : : = ⟨имя входного файла⟩$_l$

где $l \in N$. Число $l$ будем называть индексом записи.

Определим гашение: ⟨гашение⟩ : : = $\overline{\langle\text{запись}\rangle}$

Определим условие:

⟨условие⟩ : : = $\left[\begin{array}{l}\langle\text{запись}\rangle\ \langle\text{логическая операция}\rangle\ \langle\text{запись}\rangle \\ \langle\text{запись}\rangle\ \langle\text{логическая операция}\rangle\ \langle\text{константа}\rangle \\ \langle\text{константа}\rangle\ \langle\text{логическая операция}\rangle\ \langle\text{запись}\rangle\end{array}\right]$

⟨логическая операция⟩ : : = $< \mid \geqslant$

⟨константа⟩ : : = $0 \mid \pm 1 \mid \pm 2 \mid \ldots$

Пример. Если $A$ и $B$ — входные файлы, то $[A_{13} < B_{17}]$ и $[A_{17} < 1]$ есть условия.

Определим результат:

⟨результат⟩ : : = ⟨имя выходного файла⟩ (⟨запись⟩) |
⟨имя выходного файла⟩ (⟨константа⟩)

Определим условие конца файла:

⟨условие конца файла⟩::=⟨имя входного файла⟩*

Ф-словом является конечное сцепление записей, гашений, условий и результатов.

⟨Ф-слово⟩::= ⟨запись⟩|⟨гашение⟩|⟨условие⟩|⟨результат⟩|
⟨Ф-слово⟩⟨Ф-слово⟩

Определим цикл ⟨тело цикла⟩::=⟨Ф-слово⟩|⟨цикл⟩|
⟨тело цикла⟩⟨тело цикла⟩

⟨цикл⟩::=⟨·⟨тело цикла⟩·⟩

Определим формулу как конечное сцепление Ф-слов, циклов и условий конца файлов.

⟨формула⟩::=⟨Ф-слово⟩|⟨цикл⟩|⟨условие конца файла⟩|
⟨формула⟩⟨формула⟩

Элементами формулы будем называть записи, гашения, условия, результаты и условия конца файла, не являющиеся частью другого элемента.

Пример. а) $A_{17}$; б) $U(A_{17})$.

$A_{17}$ в примере а) является элементом формулы, а в примере б) не является таким, потому что она часть результата.

Определим $k_F$ — число доступных записей файла $F$ в точках формулы справа от каждого элемента. Пронумеруем все такие точки слева направо и добавим «нулевую» точку слева от первого элемента.

Тогда 1) $k_F(0)=0$.

2) Если $k_F(j)=m$ и $j+1$-й элемент формулы есть запись файла $F$, то $k_F(j+1)=m+1$.

3) Если $k_F(j)=m$ и $j+1$-й элемент формулы есть гашение файла $F$, то $k_F(j+1)=\max\{0, m-1\}$.

4) Если $k_F(j)=m$ и $j+1$-й элемент формулы не является ни записью, ни гашением файла $F$, то $k_F(j+1)=k_F(j)=m$.

Перейдем к определению ФОФ.

Формулу будем называть формулой обработки файлов, если выполняются следующие правила:

1) В формуле справа от условия конца файла не находится ни одна запись этого файла.

2) Для всех элементов в формуле индекс записи а) равен $k_F(j-1)+1$, если элемент есть запись файла $F$; б) не превосходит $k_F(j-1)$, если элемент содержит запись файла $F$ ($j$ — номер обозреваемого элемента, совпадающий с номером ближней правой от него точки).

3) Для каждого файла и каждого цикла число доступных записей файла перед циклом равно числу доступных записей после цикла.

ФОФ, не содержащие гашения и циклы, будем называть функциями пути (ФП).

### 3. СОДЕРЖАТЕЛЬНАЯ СВЯЗЬ ФОФ И ПРОГРАММ НА БАЗОВОМ ЯЗЫКЕ

Пусть нам даны программа на базовом языке и произвольный путь в ней. На этом пути программа реализует какое-то преобразование данных. Дадим алгоритм построения описания этого преобразования, представляющий собой символическое выполнение данного пути.

Будем просматривать весь путь от начала до конца, при этом фиксируя после символического выполнения каждой команды, какие записи или константы содержатся во внутренних ячейках (имя файла и порядковый номер записи). Перечислим необходимые действия при каждой возможной команде.

1) $X \Rightarrow t$ с выходом «+»; фиксируем, что в $t$ содержится $X_i$, где $i$ — количество команд этого типа для данного файла с начала пути, включая обрабатываемую; пишем $X_i$.

2) $X \Rightarrow t$ с выходом «−»; пишем $X^*$, если такого элемента в формуле еще нет. Если такой элемент есть, ничего не пишем.

3) $t \Rightarrow Y$; пишем $Y(X_i)$ или $Y(C)$ в зависимости от того, что в данный момент содержится в $t$.

4) $v \Rightarrow t$ $(C \Rightarrow t)$; ничего не пишем, отмечаем, что в $t$ содержится то же, что и в $v(C)$.

5) $t<v$ $(t<C)$ и выход по «+»; пишем $\lfloor X_i<Z_j \rfloor$ $(\lfloor X_i<C \rfloor)$, где $X_i$ и $Z_j$ — записи, содержащиеся в $t$ и $v$ (если $t$ или $v$ содержит константу, записывается константа).

6) $t<v$ $(t<C)$ и выход по «−»; пишем $\lfloor X_i \geqslant Z_j \rfloor$ $(\lfloor X_i \geqslant C \rfloor)$ или вместо обозначений записываем константы, содержащиеся в ячейках.

7) СТОП — завершаем построение.

Примечание. Если при обработке условия обе ячейки (единственная ячейка) содержат константы, то полученное неравенство ложно или истинно. Если оно ложно, то функция на данном пути не определена и на этом завершаем ее построение. Если же неравенство истинно, ничего не пишем.

Из алгоритма следует, что нами получена ФП.

Дадим пример ФП: $A_1B_1\lfloor A_1<B_1 \rfloor U(A_1)A^*U(B_1)B^*$.

Эта функция соответствует преобразованию данных на пути (*1—8*) программы на рис. 2. Последовательности, соответствующие файлам $A$ и $B$, должны содержать в точности одно число, а последовательность, соответствующая файлу $U$, — в точности два числа. При этом число последовательности, соответствующей файлу $A$, должно быть меньше числа последовательности, соответствующей файлу $B$. Значением частичной функции, определенной данной ФП, на паре последовательностей $\{1\}$ и $\{2\}$ является последовательность $\{1, 2\}$. На паре $\{1\}$ и $\{0\}$ эта функция не определена.

Если бы удалось каким-то образом объединить ФП всех путей программы (а их, как правило, неограниченное число), то

была бы получена функция программы. Для этого нужно стянуть «повторяющиеся» части различных ФП в «циклы». Именно это и делается при помощи ФОФ. Теперь понятными должны стать формальные ограничения в определении ФОФ:

Запрещается чтение файла после его окончания.

Разрешается чтение только следующей непрочитанной записи и запрещается использование записей, не находящихся во внутренних ячейках. Число доступных записей фактически соответствует числу внутренних ячеек программы.

Далее будет показано, в каком смысле ФОФ объединяет множество функций путей, а именно — представлена процедура генерирования из данной ФОФ конкретных ФП.

## 4. ПРОЦЕДУРА ГЕНЕРИРОВАНИЯ ФП, ОСНОВАННАЯ НА ФОФ

Процедура генерирования ФП (недетерминированная) состоит из двух частей — разворачивания и индексирования.

### 4.1. Разворачивание

Р1) Если данная ФОФ не содержит циклов, то переходим к индексированию, если содержит, то переходим к Р2.

Р2) Находим в ФОФ первый слева цикл. Изменяем ФОФ, на место этого цикла записываем $k$ раз его тело, где $k$ — произвольное неотрицательное целое число. Переходим к Р1.

Примечание. Можно доказать, что полученная после разворачивания формула также есть ФОФ.

### 4.2. Индексирование

Для этой части нам понадобится таблица доступных записей вида (⟨запись⟩, ⟨абсолютный номер⟩). Число элементов этой таблицы не превышает $\max \sum_{1 \leqslant j \leqslant mF \in V} k_F(j-1)$ (где $m$ — число элементов в ФОФ, $V$ — множество имен входных файлов). Будем использовать также таблицу входных файлов (⟨имя входного файла⟩, ⟨номер последней прочитанной записи⟩).

Будем обрабатывать формулу слева направо по алгоритму:

И1. Берем следующий элемент формулы.

И2. Если это запись, переходим к И3.
Если это гашение, переходим к И4.
Если это условие, переходим к И5.
Если это результат, переходим к И6.
Если это условие конца файла, переходим к И1.

И3. Если файл не занесен в таблицу файлов, заносим его туда и ставим номер последней прочитанной записи 0. Увеличиваем номер последней прочитанной записи файла на один. В таблицу доступных записей заносим запись с абсолютным номером, равным номеру последней прочитанной записи файла. В обозреваемом элементе ФОФ заменяем индекс записи на абсолютный номер записи из таблицы доступных записей. Возвращаемся в И1.

И4. В таблице доступных записей находим запись, которая соответствует данному гашению, и стираем ее из таблицы. Для всех записей данного файла с индексом, большим стертого, уменьшаем индекс записи на единицу. Выбрасываем обозреваемый элемент из ФОФ. Возвращаемся в И1.

И5. Для обеих записей (если условие содержит две записи) или для одной записи в таблице доступных записей находим их абсолютные номера и ставим эти номера вместо индексов записи обозреваемого элемента ФОФ. Возвращаемся в И1.

И6. В таблице доступных записей находим запись, являющуюся частью результата, и в элементе ФОФ заменяем индекс записи на ее абсолютный номер. Возвращаемся в И1.

Содержательно в части индексирования происходит переход от относительных номеров записей входных файлов к их абсолютным номерам. Для этого необходимы связь между относительными номерами записей и их абсолютными номерами (таблица доступных записей) и число уже прочитанных записей каждого файла. Можно доказать, что после обработки $j$-го элемента формулы подформула, содержащая с первого по $j$-й элемент формулы, является ФОФ. Из этого следует, что после окончания работы алгоритма нами будет получена ФП.

## 5. ОСНОВНАЯ ТЕОРЕМА

Теорема. Существует алгоритм $A$, который для каждой программы П на базовом языке генерирует конечную систему формул обработки файлов $A(\text{П})$, задающую то же преобразование, что и сама программа П.

*Доказательство* теоремы аналогично доказательству основной теоремы в [4]. Из-за громоздкости оно здесь не приводится.

## 6. ПРИМЕР ФОФ

В качестве примера приведем систему ФОФ, построенную для программы (рис. 2) из [1].

1. $A_1B_1 < \cdot < \cdot \lfloor A_1 < B_1 \rfloor U(A_1)\bar{A}_1A_1 \cdot > < \cdot \lfloor A_1 \geqslant B_1 \rfloor \lfloor B_1 < A_1 \rfloor U(B_1)\bar{B}_1B_1 \cdot > \cdot > \lfloor A_1 < B_1 \rfloor U(A_1)A^*U(B_1)\bar{B}_1B_1 < \cdot U(B_1)\bar{B}_1B_1 \cdot > U(B_1)B^*$

2. $A_1B_1 < \cdot < \cdot \lfloor A_1 < B_1 \rfloor U(A_1)\bar{A}_1A_1 \cdot > < \cdot \lfloor A_1 \geqslant B_1 \rfloor \lfloor B_1 < A_1 \rfloor U(B_1)\bar{B}_1B_1 \cdot > \cdot > \lfloor A_1 < B_1 \rfloor U(A_1)A^*U(B_1)B^*$

3. $A_1B_1 < \cdot < \cdot \lfloor A^1 < B_1 \rfloor U(A_1)\bar{A}_1A_1 \cdot > < \cdot \lfloor A_1 \geqslant B_1 \rfloor \lfloor B_1 < A_1 \rfloor U(B_1)\bar{B}_1B_1 \cdot > \cdot > \lfloor A_1 \geqslant B_1 \rfloor \lfloor B_1 < A_1 \rfloor U(B_1)B^*U(A_1)\bar{A}_1A_1 < \cdot U(A_1)\bar{A}_1A_1 \cdot > U(A_1)A^*$

4. $A_1B_1 < \cdot < \cdot \lfloor A_1 < B_1 \rfloor U(A_1)\bar{A}_1A_1 \cdot > < \cdot \lfloor A_1 \geqslant B_1 \rfloor \lfloor B_1 < A_1 \rfloor U(B_1)\bar{B}_1B_1 \cdot > \cdot > \lfloor A_1 \geqslant B_1 \rfloor \lfloor B_1 < A_1 \rfloor U(B_1)B^*U(A_1)A^*$

5. $A_1B^* < \cdot U(A_1)\bar{A}_1A_1 \cdot > U(A_1)A^*$

## 7. ЗАКЛЮЧЕНИЕ

Понятно, что вышеизложенные ФОФ и процедура генерации конкретных ФП и ФОФ применимы только для весьма узкого класса преобразования данных. Это преобразования последовательных файлов, не включающие сложные преобразования полей: арифметику, сцепление и т. п. Выбор ФОФ и процедуры обусловлен нашим желанием выявить тот язык программирования, для которого:

1) существует алгоритм автоматического построения ФОФ по тексту программы;

2) построенная система ФОФ однозначно определяет функцию программы, т. е. из совпадения ФОФ для двух программ следует их функциональная эквивалентность и, кроме того, по исходным данным при помощи только ФОФ определяется результат работы программы.

## СПИСОК ЛИТЕРАТУРЫ


1. Барздинь Я. М., Бичевский Я. Я., Калниньш А. А. Построение полной системы примеров для проверки корректности программ. — Учен. зап. ЛГУ им. П. Стучки, 1974, т. 210, с. 152—187.
2. Бичевский Я. Я. Автоматическое построение полных систем примеров. Дис. на соиск. учен. степ. канд. физ.-мат. наук. Рига, 1977. 112 с
3. Бичевский Я. Я., Борзов Ю. В. К вопросу о формальном описании функции, реализуемой программой. — В кн.: Проблемы проектирования и применения дискретных систем в управлении. Тез. докл. Первой междунар. конф. молодых ученых. Минск, 1977, с. 207—208.
4. Борзов Ю. В. Автоматическое построение описания функции, реализуемой программой ЭВМ. Латв. гос. ун-т, Рига, 1980. 25 с. Рукопись деп. в ВИНИТИ 25.02.81, № 887-81. Деп.
5. Ефимкин К. Н., Задыхайло И. Б. О верификации программ на одном языке. — Программирование, 1980, № 2, с. 69—77.
6. Boyer R. S., Elspas B., Levitt K. N. SELECT — a formal system for testing and debugging programs by symbolic execution. — In: Proc. 1975 Intern. Conf. reliable software. Los Angeles, 1975, p. 234—245.
7. Howden W. E. Symbolic testing and the DISSECT symbolic evaluation system. — IEEE Trans. software eng., 1977, SE-3, N 4, p. 266—278.
8. King J. C. A new approach to program testing. — In: Proc. 1975 Intern. Conf. reliable software. Los Angeles, 1975, p. 228—233.



*Вычислительный центр*
*ЛГУ им. П. Стучки*



*Поступила 18.12.81*

M.I.Nurminen, P.Järvinen, G. Weir (eds.):
Precedings of Human Jobs and Computer
Interfaces Conference, pages 471-475
University of Tampere, Finland, 1991



# THE WAY LATVIAN ERGONOMIC KEYBOARD WAS DESIGNED

**Juris Borzovs**
Research Institute of Mathematics and Computer Science
The University of Latvia

The pressure of Russian language, some subtle russification tendences have lead Latvian language to a very hard and specific situation, namely, nearly every document was written in Russian, official conversation also mainly was taken in Russian, etc.

It is not a surprize, therefore, that the standard configuration of Latvian keyboard was adopted not earlier than in 1985. Unfortunately, it was a very rough and formal adaptation of well-known English-American "qwerty" keyboard. Naturally, this standard did not consider specific aspects of Latvian language, it was not ergonomic one (meanwhile, "qwerty" is not ergonomic one even for English). Rather narrow stream of Latvian texts did not require substantial amount of Latvian typewriters and therefore the meant keyboard standard was really "unknown", it did not "work" in everyday life.

This situation was radically changed in 1989, when Latvian parliament adopted Latvian language as the state language (now usage of Latvian and Russian is equally obliged in government, official documents, service, such as shopping, medicine, police, etc.). Of course, stream of Latvian texts is going up now, but nonadequacy of the keyboard cuts down potential speed of typewriting (including personal computers and desk-top publishers) up to 2-3 times. The point is that today we have extremely low level of equipment for typewriting in Latvian and this yields us unique possibility to change the keyboard with minimal impact on users.

## 1. The characteristics of "qwerty"-like Latvian keyboard

Two methods of typewriting are used in praxis - "looking" method and "ten-fingers" method. The first one is used by non-professionals, the second one is mainly used by professional typists. The fundament of "ten-fingers" method is a division of keys into fixed zones: every finger serves definite zone of keys (Fig.1). If somebody works using "ten-fingers" method the effectiveness of typewriting is substantially affected by location of keys (and letters) on keyboard. Incorrect location of keys cuts down effectiveness of typists job.

Latvian typewriter being in use now is designed on the basis of English "qwerty" standard keyboard. The lacking letters was simply added in periphery of keyboard (entering of the letters "k with cedilla","l with cedilla","n with cedilla" and "g with cedilla above" is realized by the aid of special "dumb" key yielding a "sign of softening": it takes pressing of two keys to type these letters)(see Fig.1 and Fig.2). Keyboard adapted in such a way has the following deficiences.

1) It predicts nonrational location of Latvian alphabet letters yielding substantial cut of job effectiveness when typist uses "ten-fingers"("blind") method. E.g.,the letters "f" and "h" very seldom

appearing in Latvian texts are located in the centre of keyboard and printed by forefingers. Contrary, several often used letters are located at the margines (the most frequently used latvian character "a" is printed by little finger of the left hand, the third letter "s" - by ring-finger of the left hand, the nineth character "a with macron" - by little finger of the right hand). Writing latvian text on such keyboard about 50% of characters are entered by the second and the third finger, although English text on "qwerty" yields about 70%. Fig.2 demonstrates frequency of usage of latvian characters gathered from sample texts of 60 000 letters (blank between words also counted as a letter).

Present latvian keyboards give too great load to the fourth and the fifth finger. Already during the existance of the first Republic of Latvia (Latvijas Republika) in 1918-1940 there was found that using any latvian keyboard weeker fingers (ring- fingers and little fingers) are overloaded but stronger fingers (forefingers and middle fingers) works with less load. This is one of conditions because some typists graduated from courses begin to type using nonrational "looking" method - their hands are overloaded, soon get tired and sometimes even get pain, particularly palms around ring-fingers and little fingers. The result is that average speed of typewriting is 80-100 letters per minute contrary 300-400 letters per minute abroad. Therefore, installation of typewriters to answer requirements of ergonomics would increase efficiency of typist job 3-4 times.

An obvious defect of present latvian typewriters is usage of common keys for digit "1" and letter "l", digit "0" and letter "O". There follows that such keyboards can not be used for personal (and other) computer.

Very specific defect is lack of Latgallian (the language being used in the Eastern part of Latvia) characters, particularly "o with macron".

## 2. Criteria of ergonomics of Latvian keyboard

To design and evaluate new ergonomic keyboard the following criteria were adopted.

Working load of fingers. Forefingers and middle fingers are stronger and mobiler than ring-fingers and little fingers. Therefore letters should be located on keyboard so that as much as possible part of text could be entered by forefingers and middle fingers.

Moving of fingers from the basic position. Using "ten-finger" method hands are constantly located on definite, one and the same part of keyboard. This location is called basic position. Thus exactly eight characters can be entered without moving of finger from the basic position (using 2th-5th fingers of the both hands). To enter other characters move corresponding finger from basic position, press necesarry key and return the finger into the basic position. For this reason, entering of characters not located in the basic position requires additional time and working load. This additional time and working load is different for different keys. Taking into consideration this factor and the fact that fingers differs in strength and mobility, every key can be assigned some estimation

number - coefficient of hardness. To obtain coefficients of hardness professional typists were interviewed (of course, such procedure yields rather subjective results). Obtained coefficient of hardness shows how many times harder is entering of character by given key compared with the basic position of forefinger. The coefficients of hardness are demonstrated on Fig.1 under corresponding keys.

Rhytm. It was found that a typist gets tired less and works quicker when the text contains more pairs of character whose entering demands changing of hands. It can be assumed that the hardness of entering of a character depends also on the fact the previous character was entered by the same or the other hand. We assume that the hardness changes also depending on fact previous character was entered by the same or some other finger.

Learnability. When somebody learns to type on typewriter or computer he/she progresses faster if texts are meaningful even then if he/she types only several characters. E.g., "roll,pool,loop,..." - combinations of p,r,o,l. To state the fact, the "qwerty" is not convenient for this reason.

## 3. Design activities

To automate some labour-consuming parts of the project a computer program was created. This IBM PC programm (written in Prolog) inputs keyboards' configuration, initial location of characters, coefficients of hardness and subsequently outputs keybords' evaluation characteristics. It also has a special feature to seek for the best letter location if the set of all characters is given.

Fig.3 demonstrates the new Latvian-Latgallian-English ergonomic (naturally, only for Latvian) keyboard. Numbers written under keys show frequences of letters. The keyboard contains 29 of 33 letters of Latvian alphabet ( the rest "k with cedilla", "l with cedilla", "n with cedilla" and "g with cedilla above" are entered by the aid of special "dead" key coupled with subsequent pressing of "k","l","n" or "g"), Latgallian "o with macron", all English letters and all decimal digits.

Fig.4 demonstrates comparison of the new keyboard and several other keyboards being used in Latvia.

## Acknowledgements

The author express his gratitude to Dr.Juris Raats and Mr.Andris Kalnroze for their invaluable contribution in this project. Dr.Juris Raats created the computer program, Mr.Andris Kalnroze designed the final version of the new ergonomic keyboard.

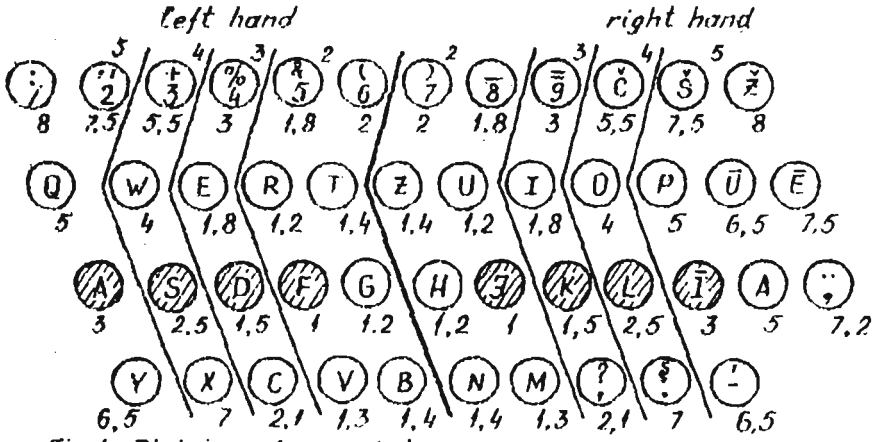



Fig. 1. Division of keys into zones, coefficients of hardness, basic position of fingers

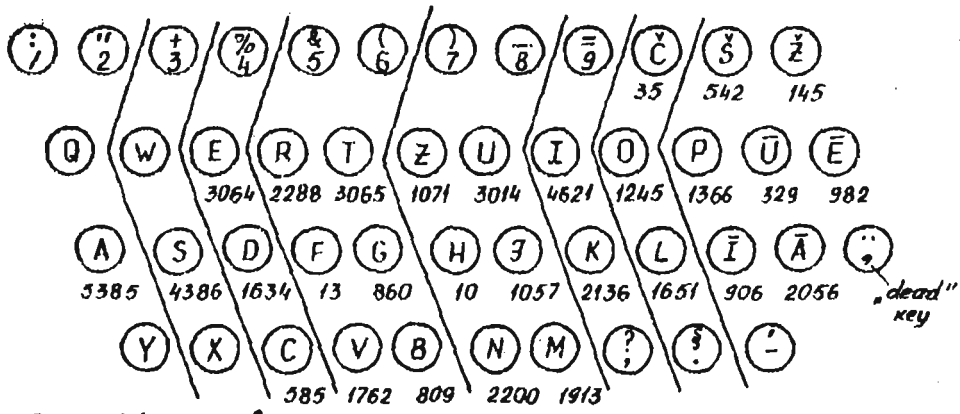



Fig. 2. Working load of fingers (frequency of letters) on Latvian qwerty

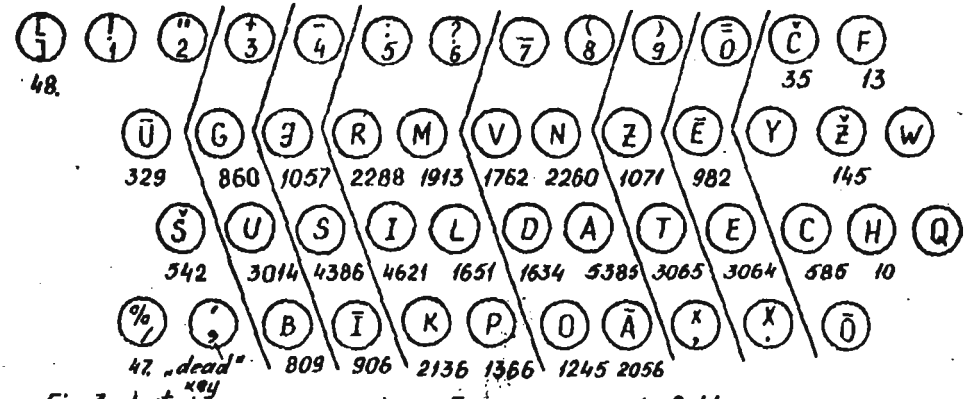



Fig. 3. Latvian ergonomic. Frequency of letters

| Keyboard | Entered by 2nd and 3rd fingers | Entered in basic position | Entered changing hands | Evaluated tiredness of hands |
|---|---|---|---|---|
| Kontinental | 67.9% | 43.6% | 67.5% | 2.19 |
| Mercedes | 67.9% | 43.6% | 67.6% | 2.16 |
| Remington | 68.4% | 43.6% | 71.9% | 1.92 |
| Royal | 67.9% | 42.4% | 68.5% | 2.29 |
| Underwood 2 | 67.9% | 42.2% | 68.0% | 2.44 |
| Latvian qwerty | 67.9% | 43.6% | 67.5% | 2.57 |
| Latvian ergon | 80.2% | 58.1% | 73.4% | 1.73 |

Fig. 4. Comparing of keyboards being in use in Latvia

АКАДЕМИЯ НАУК СССР

# АВТОМАТИКА
# И
# ТЕЛЕМЕХАНИКА

(ОТДЕЛЬНЫЙ ОТТИСК)



8

МОСКВА · 1982

УДК 681.3.06

# РАЗВИТИЕ МЕТОДОВ СИМВОЛИЧЕСКОГО ТЕСТИРОВАНИЯ ПРОГРАММ ЭВМ

БИЧЕВСКИЙ Я. Я., БОРЗОВ Ю. В.

(Рига)

Рассматривается применение символического тестирования для проверки правильности программ ЭВМ: автоматического генерирования входных данных (тестов), полных систем тестов, выделения всех реализуемых путей программы, генерирования описания функции, которую реализует программа. Приводятся различные описания функции, реализуемой программой, отличающиеся полнотой ее задания.

## Введение

Основным способом выявления правильности программ ЭВМ по сей день остается тестирование, т. е. выполнение вычислений на различных входных значениях (тестах). Принято считать, что тестированием невозможно доказать правильность программы (кроме случая, когда программа выполняется на всех возможных входных значениях), однако на практике при достаточно большом количестве тестов, на которых программа выдает правильные результаты, уверенность в ее правильности становится достаточно большой. Чем больше известно о выполняемых программой преобразованиях данных, тем выше эта уверенность.

Рассмотрим совокупность методов автоматического тестирования программ ЭВМ, составляющих в некотором смысле определенный подход. Этот подход характеризуется иерархией методов, дающих все более полное представление о функции, которую вычисляет программа, а также использованием символического выполнения программы в качестве основного инструмента тестирования (символического моделирования — в терминологии [1]; символического исполнения — в терминологии [2]). Будет представлен спектр этих методов, начиная с несложных и достаточно давно известных — генерирования числового теста, проходящего заданный путь программы, и функции пути (разделы 1 и 2); весьма нетривиальных — генерирования полной системы тестов и системы функции покрывающего множества путей (разделы 4 и 5); и кончая автоматическим выделением всех реализуемых путей программы и генерированием функции программы (раздел 6). Первая группа методов дает относительно небольшую информацию о работе программы, тогда как вторая и третья группы дают все возрастающий объем информации о производимых программой преобразованиях данных.

Если методы первых двух групп уже используются в промышленных или по крайней мере экспериментальных системах автоматического тестирования программ [3], то методы третьей группы (насколько это известно авторам) в данной статье предлагаются впервые.

## 1. Генерация теста для отдельного пути

Тестирование программы можно осуществить следующим образом. Выберем из программы наиболее характерные пути программы и для них построим тесты, на которых эти пути выполняются. Нахождения теста для выбранного пути можно проводить как вручную, так и автоматически. Ясно, что этот метод не гарантирует правильность работы программы на всех возможных входных значениях даже в том случае, если программа выдала

правильные результаты на выбранных тестах. Однако, как отмечалось, уверенность в правильности программы в случае успешного тестирования значительно возрастает.

Предположим, что дан путь (1, 2, . . . , 6):

```
13      READ 5, X, Y                1
        IF (X.LE.Y) GO TO 17        2
        Z=X                         3
        X=Y                         4
        Y=X                         5
17      X=Y-X                       6
        IF (X.LE.0) GO TO 13        7
```

и надо найти входные значения (тест), при которых этот путь будет выполнен.

Попробуем выполнить данный путь на символических значениях. Это означает, что оператор 1 присваивает переменным X и Y символические значения, например $X_1$ и $Y_1$ соответственно. Оператор 2 сравнивает значения переменных X и Y. Если значение X меньше или равно значению Y (т. е. $X_1 \leqslant Y_1$), то управление будет передано на оператор 6. После выполнения операторов 3, . . . , 6 переменные будут содержать следующие символические значения: $X=0$, $Y=Y_1$, $Z=X_1$, причем между значениями установится отношение $X_1 > Y_1$.

Теперь, чтобы генерировать тест, на котором программа пройдет данный путь, достаточно решить неравенство (в общем случае систему неравенств) $X_1 > Y_1$. Более точно, надо отыскать хотя бы одно ее решение, например $\{x_1=2, y_1=1\}$. Пара чисел (2,1) и будет тем тестом (входными данными), на котором выполняется путь (1, 2, . . . , 6).

Ясно, что из-за отсутствия общего метода решения системы неравенств любой сложности также невозможна генерация теста для прохождения любого пути. В этой связи большой интерес представляют быстродействующие частичные алгоритмы решения систем неравенств, используемые в машинных системах генерации тестов [4—7].

## 2. Генерация функции пути

На каждом пути программы происходит некоторое преобразование данных (быть может, пустое). Следовательно, каждый путь определяет функцию (быть может, нигде не определенную, если путь нереализуем) — так называемую функцию пути. Рассмотренный выше метод символического выполнения пути фактически дает возможность выявить и аналитическую запись такой функции. Если воспользоваться предыдущим примером, то путь (1, 2, . . . , 6) задает функцию, преобразующую пару чисел $(X', Y')$ в тройку чисел $(X'', Y'', Z'')$, где $X''=0$, $Y''=Y'$, $Z''=X'$, причем функция определена для всех таких пар $(X', Y')$, для которых выполняется условие $X' > Y'$.

Еще нагляднее функцию пути можно продемонстрировать на следующем примере, взятом из [8]:

```
      DOUBLE PRECISION FUNCTION SIN(X)
      DOUBLE PRECISION E, TERM, SUM
      REAL X
      TERM=X
      SUM=X
      DO 20 I=3, 100, 2
      TERM=TERM*X**2/(I*(I-1))
      SUM=SUM+((-1)**(I/2))*TERM
      IF (DABS (TERM). LT. E) GO TO 30
20    CONTINUE
```

```
30      SIN=SUM
        RETURN
        END
```

Если переменной X присвоить символическое значение x (переменной E — значение e) и выполнить путь, проходящий цикл дважды, SIN получит в качестве результата выражение

```
x-x**3/6+x**5/120,
```

что является записью функции данного пути. Область определения функции задает система неравенств, полученная при символическом выполнении:

```
DABS(z**3/6).GE.e
DABS(x**5/120).LT.e
```

Во многих случаях функция пути дает намного больше информации, чем числовая пара входного и выходного значения. Первыми автоматическими системами генерации функции пути по заданному пользователем пути являются [8—12].

### 3. Определение достаточности тестирования

Как известно, полное тестирование программы (т. е. ее тестирование на всех возможных входных значениях) невозможно из-за огромного (практически бесконечного) числа вариантов. В случае обработки файлов их длина, как правило, ничем не ограничивается, что порождает неограниченность числа входных данных и путей программы, так как пути, отличающиеся лишь числом прохождения цикла,— различные пути. Поэтому необходимо ограничить количество тестов до разумного предела, что в свою очередь требует установления определенных критериев такого ограничения. Перечислим ряд таких критериев, основанных на управляющем графе программы:

1) тестирование всех путей;
2) тестирование всех ветвей из всех возможных состояний [13, 14];
3) тестирование всех таких путей, в которых итерации циклов, зависящих от входных данных, ограничены некоторой глобальной константой N (чаще всего N=2);
4) тестирование всех ветвей;
5) тестирование всех операторов.

Наиболее часто применяются критерии 3) и 4). В системах [6, 7, 15, 16] предполагается, что программа тестирована достаточно, если тесты удовлетворяют критерию 4). Совокупность тестов, удовлетворяющая 4), называется полной системой тестов [17].

### 4. Генерация полной системы тестов

Автоматическая генерация полной системы тестов предполагается в [6, 7, 15, 18—21] и др. Однако только в [7] используется метод [14, 17], дающий возможность хотя бы в принципе выделить подмножество языка программирования, для которого генерация полной системы тестов гарантирована.

Рассмотрим обычно используемые фазы генерации тестов, в основном опираясь на опыт [7].

*а. Определения и внутреннее представление программы.* Большинство автоматических систем использует внутреннее представление программы в виде управляющего графа. Вершины графа представляют команды программы, а ориентированные дуги — возможные передачи управления. Пути и ветви программы обычно определяются исходя из соответствующего управляющего графа программы [17, 22] G(V, A), где V есть множество вершин и A — множество ориентированных дуг.

Путем программы назовем такую последовательность вершин и дуг в графе ($V_1$, $A_{1,2}$, $V_2A_{2,3}$, ..., $A_{k-1,k}$, $V_k$), где каждая дуга $A_{1,1+1}$ выходит из вершины $V_1$ и входит в вершину $V_{1+1}$.

Ветвью программы назовем такой путь ($V_1A_{1,2}$, $V_2$, $A_{2,3}$, ..., $A_{k-1,k}$, $V_k$), где $V_1$ — либо первая команда программы, либо условная команда; $V_k$ — либо условная команда, либо команда выхода из программы; все остальные $V_1$ ($1<i<K$) — безусловные команды.

Если существуют входные данные, на которых программа выполняет данный путь (ветвь), этот путь (ветвь) называется реализуемым.

Итак, первая фаза генерации тестов обычно состоит в переводе исходного текста анализируемой программы в управляющий граф. Одновременно проводится так называемое статическое тестирование программы, представляющее собой выявление свойств программы без какого-либо ее выполнения на данных.

Здесь выявляются ошибки типа: недостижимый оператор, неинициализированная переменная, присвоение значения переменной без его последующего использования и т. п.

Однако следует еще раз подчеркнуть, что эти ошибки, возможно, вовсе не повлияют на работу программы, так как могут быть связаны с нереализуемыми путями. На этой фазе фактически применяются известные методы анализа программы [23, 24].

*б. Выделение покрывающего множества путей.* Вторая фаза генерации тестов предполагает выделение подмножества путей программы, содержащих в совокупности все дуги управляющего графа,— покрывающего множества путей. Большинство известных систем выделяют такое подмножество статическим анализом управляющего графа [6, 18] или представляют выбор путей пользователю [10, 19] [1]. Ясно, что так выбранные пути вполне могут оказаться нереализуемыми (и задача генерации полной системы тестов — невыполненной). В системе [7] предложен существенно другой метод. Последовательным просмотром управляющего графа программы [2] у каждой вершины графа отмечается, какие значения перед выполнением соответствующей команды могут в принципе содержать все переменные программы. Это описание области возможных значений переменных [3] называется состоянием. Если оказывается, что у некоторой вершины хотя бы одна переменная имеет пустое множество возможных значений, ясно, что соответствующая команда принадлежит нереализуемой ветви. Если программа не содержит сложных арифметических преобразований данных, то обычно таких описаний оказывается конечное число.

Фактически производится символическое выполнение программы. Для каждой команды программы и приписанного к ней состояния выясняется, какой команде будет передано управление и как при этом изменится состояние, приписанное к этой последней команде. Такой анализ пар (команда, состояние) проводится до тех пор, пока получается полное покрытие управляющего графа или же выясняется, что из-за нереализуемости некоторой ветви программы полное покрытие получить невозможно.

Допустим, что к команде $K_i$ приписаны состояния $S_1(K_i)$, $S_2(K_i)$, ... ..., $S_n(K_i)$, что содержательно означает: выполняя различные пути, приводящие в команду $K_i$, переменные программы могут содержать значения, в одном случае описываемые состоянием $S_1(K_i)$, в другом — $S_2(K_i)$ и т. д., наконец, $S_n(K_i)$. Предположим, что анализируем передачу управления из $K_i$ в $K_{i+1}$. Тогда следует выяснить, как меняются значения переменных (состояния $S_1(K_i)$, $S_2(K_i)$, ..., $S_n(K_i)$) при выполнении команды $K_i$ (с переходом на $K_{i+1}$), и приписать соответствующие состояния к $K_{i+1}$. Вполне

---

[1] Отметим также метод естественного выделения покрывающего множества с использованием стохастического тестирования [1]. В этом случае предварительного выделения покрывающего множества путей не происходит.

[2] Порядок просмотра управляющего графа не является существенным.

[3] Состояние может содержать описания областей значений всех переменных программы, хотя на практике в состояние включаются лишь те переменные, значения которых влияют на передачу управления.

возможно при этом, что придется к $K_{i+1}$ приписать новые состояния, которые до сих пор нигде при анализе не встречались. Также возможно, что новые состояния, кроме уже приписанных к $K_{i+1}$, не образуются. Может оказаться, что вновь приписываемое состояние описывает пустое множество допустимых значений (пустое состояние); это значит, что обнаружена нереализуемая при данных значениях ветвь. Если случается, что все состояния, приписанные к команде, — пустые, то ветвь программы нереализуема ни при каких входных значениях.

Вообще говоря, такой анализ программы может продолжаться бесконечно, однако в задачах обработки данных (для которых этот метод изобретен) количество различных состояний часто не превосходит нескольких десятков.

В теоретических исследованиях [14, 17] используемый язык программирования таков, что конечное число различных состояний гарантируется для любой программы на этом языке.

Описанное построение легче всего представить как некоторое дерево. Его вершинами служат пары вида («номер команды $K_i$», «состояние $S_i(K_i)$»), а дугами обозначаются допустимые переходы из команды $K_i$ и состояния $S_m(K_i)$ в команду $K_{i+1}$ и состояние $S_n(K_{i+1})$. Это дерево называется деревом реализуемости [17]. Для базового языка [17] любая реализуемая ветвь программы содержится в дереве реализуемости, и оно содержит только реализуемые ветви.

Теперь для выделения путей, содержащих в совокупности все реализуемые ветви программы, достаточно выбрать любое покрытие дерева реализуемости путями, начинающимися с первой и оканчивающимися командой выхода из программы.

Следует подчеркнуть, что именно реализуемость всех выбранных путей существенно отличает данный метод от других.

Рассмотрим в качестве примера состояния программы, представленной в разделе 1. После выполнения оператора 1: $X=x_1$, $Y=x_2$, где $x_1$ и $x_2$ — символические значения, считываемые оператором READ (Z не включена в состояние, так как ее значения на передачу управления не влияют).

После выполнения пути $(1, 2, \ldots, 6)$:

$X=x_2-x_2$, $x_1>x_2$, $Y=x_2$.

После выполнения пути $(1, 2, \ldots, 7, 1)$:

$X=x_3$, $Y=x_4$.

Отношение $x_1>x_2$ исключено, так как ни одна переменная, влияющая на передачу управления, уже не содержит значений $x_2$ и $x_1$. Первое и третье состояния совпадают с точностью до обозначений символических значений, что означает равенство состояний.

*в. Генерация конкретных значений тестов.* Для выбранного подмножества путей генерируются конкретные значения тестов по методу, описанному в разделе 1.

*г. Выполнение программы на тестах.* По крайней мере, в [7] имеется возможность выполнения программы на тестах с последующей распечаткой результатов.

## 5. Генерация системы функций покрывающего множества путей

Выше отмечалось, что каждому реализуемому пути программы соответствует функция пути — отображение некоторого подмножества области определения программы в некоторое подмножество области значений программы. Такая функция пути является подфункцией отображения данных, которое реализует программа. Таких подфункций чаще всего оказывается бесконечно много, так как в большинстве случаев программы имеют неограниченно много различных путей.

Если предположить, что реализуемые пути, по которым строится полная система тестов, достаточно хорошо характеризует функционирование программы, то можно также предположить, что объединение соответствую-

щих этим реализуемым путям функций путей достаточно хорошо характеризуют функцию программы.

Системой функций покрывающего множества путей назовем систему таких функций реализуемых путей программы, которые в совокупности содержат все реализуемые ветви программы.

Прототипными системами генерации системы функций покрывающего множества путей можно с некоторыми оговорками считать [8—12, 18, 19]. С другой стороны, алгоритмы систем [6, 7] вполне пригодны для такой генерации.

Автоматическая генерация системы функций покрывающего множества гарантируется для базового языка [17].

## 6. Функция программы

Функцией программы назовем объединение функций всех путей программы.

Ясно, что отображение входных данных в выходные данные, реализуемое программой, совпадает с функцией программы.

Правильная программа — понятие относительное. Это такая программа, чья функция программы удовлетворяет спецификациям решаемой задачи. Целью отладки программы является достижение соответствия функции программы спецификациям задачи. Поэтому ясно, что получение (автоматическое) удобно читаемой функции программы для последующей проверки соответствия спецификациям решаемой задачи есть дело первостепенной важности.

В качестве записи функции программы может рассматриваться сам текст программы. Однако такой записи присущи следующие недостатки: отображение входных данных в выходные данные скрытое, неявное; нельзя узнать, принадлежат конкретные входные данные области определения функции программы или нет.

*а. Выделение множества всех реализуемых путей программы.* Одной из форм записи функции программы можно считать конечное описание всех реализуемых путей программы. Хотя и в этом случае отображение входных данных в выходные данные по всей вероятности останется неявным, однако принадлежность конкретных входных данных области определения функции программы станет разрешимой.

Понятно, что в общем случае выделение множества всех реализуемых путей программы представляет собой алгоритмически неразрешимую проблему. Но для базового языка [17] эта проблема разрешима. Так как программы обработки данных часто по своим свойствам близки базовому языку, то и во многих практических случаях возможно полное или частичное выделение интересующего множества. В системе [7] такое выделение фактически уже делалось, хотя и не во всех случаях, так как там преследовалась только цель покрытия управляющего графа программы реализуемыми путями. В настоящее время в ВЦ Латвийского государственного университета им. П. Стучки создается автоматическая система выделения множества всех реализуемых путей и генерации функций программ типа обработки данных, написанных на языке программирования ПЛ/1.

*б. Генерация функции программы.* После того как выделено множество всех реализуемых путей программы, можно заняться наглядным описанием функции программы. Такое описание должно отличаться от текста самой программы отсутствием управляющих структур и переменных, которые присущи методам реализации алгоритмов, но вовсе не являются необходимой принадлежностью отображения входных данных в выходные данные. Такое описание должно использовать понятия данных, такие, как файл, запись, поле и т. п., а также понятия операций над данными — пересылка, ввод, вывод, арифметические операции и т. д.

Тот факт, что автоматическую генерацию подобного описания функции программы удается гарантировать для того же языка, для которого разрешима проблема автоматического построения полной системы тестов [25], вселяет надежду на получение практически пригодных алгоритмов.

## 7. Примеры

Для простого и наглядного сравнения полной системы тестов, системы функций покрывающего множества путей и функции программы продемонстрируем эти три понятия на простейшей программе, написанной на базовом языке (см. рис. 1).

1. Полной системой тестов может быть, например,

$$X=(1, 1),\ U=(1, 1).$$

Файл X состоит из двух записей — чисел 1. Файл U состоит тоже из двух записей — чисел 1.

*Примечание.* Ветвь 1, 4, 5 нереализуема из-за неинициализированной переменной t.

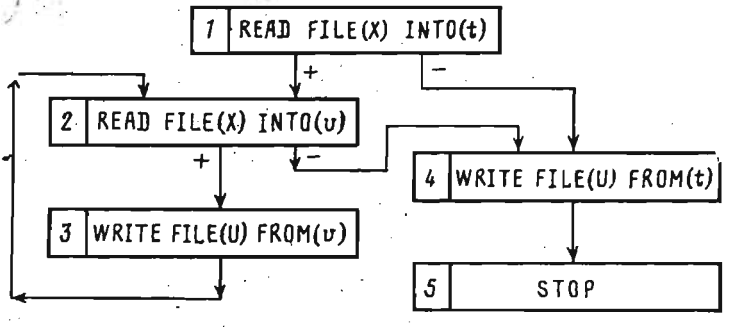



Рис. 1

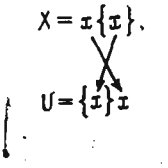



Рис. 2

Более удачная полная система тестов была бы, например,

$$X=(1, 2),\ U=(2, 1).$$

2. Система функций покрывающего множества путей:

$$X=(x_1, x_2),\ U=(x_2, x_1),$$

где $x_1$ обозначает первую запись файла X; $x_2$ — вторую.

3. Функция программы приведена на рис. 2, где {x} означает последовательность конечного числа записей файла X, в том числе и нулевого количества.

Даже такой простейший пример показывает, насколько в разной степени характеризуют программу рассмотренные три понятия.

Полная система тестов в худшем случае позволяет предполагать, что программа просто отображает входной файл из двух записей в такой же выходной файл, в лучшем случае — что при этом записи меняются местами.

Система функций покрывающего множества путей «уверенно» меняет местами записи.

И только функция программы наглядно показывает, что:

программа определена на любом файле, состоящем из конечного ненулевого количества записей;

при отображении количество записей сохраняется;

первая запись входного файла становится последней записью выходного файла, а остальные «сдвигаются», т. е. вторая запись входного файла становится первой записью выходного файла, третья — второй и т. д.

В данной статье не будем касаться известных трудностей, сопряженных с символическим выполнением программ. Многие из них описаны в [3, 26]. Эти трудности в каждом конкретном случае преодолеваются эвристическими, частными методами, рассмотрение которых потребовало бы слишком много места.

## 8. Тестирование и верификация программ

По-видимому, нет непреодолимого барьера между тестированием и верификацией.

При верификации пользователь описывает желаемую функцию и доказывает, что программа вычисляет именно ее.

При генерации функции программы (что следует считать вершиной подхода символического тестирования) по программе выявляется ее функция и затем сравнивается с желаемой функцией.

Если в первом случае при неудаче ничего нельзя сказать о том, какую же функцию программа вычисляет, то во втором даже при несовпадении функции программы и желаемой функции все-таки имеем описание фактически вычисляемой функции.

## 9. Заключение

В середине 70-х годов появился ряд экспериментальных автоматических систем отладки программ, основанных на символическом выполнении программ. На наш взгляд, в ближайшем будущем следует ожидать появления промышленных образцов таких систем, а одним из направлений развития методов тестирования будет разработка проблем изображения функции, реализуемой программой.

### ЛИТЕРАТУРА


1. *Правильщиков П. А.* Построение тестов для программ.— Автоматика и телемеханика, 1977, № 5, с. 147–161.
2. *Касьянов В. Н., Поттосин И. В.* Технологические возможности оптимизации программ.— Программирование, 1980, № 2, с. 27–31.
3. *Борзов Ю. В.* Тестирование программ с использованием символического выполнения.— Программирование, 1980, № 1, с. 51–59.
4. *Benders J. F.* Partitioning Procedures for Solving Mixed – Variables Programming Problems.— Numerische Mathematic, 1962, № 4, p. 238–252.
5. *Gomory R. E.* An Algorithm for Integer Solutions to Linear Programs.— In: Recent Advances in Mathematical Programming. N. Y.: McGraw-Hill, 1963.
6. *Ramamoorthy C. V., Ho S. F., Chen W. T.* On the Automated Generation of Program Test Data.— IEEE Trans. Software Eng., 1976, v. SE-2, № 4, p. 293–300.
7. *Bičevskis J., Borzovs J., Straujums U., Zarins A., Miller E. F., Jr.* SMOTL – A System to Construct Samples for Data Processing Program Debugging.— IEEE Trans. Software Eng., 1979, v. SE-5, № 1, p. 60–66.
8. *Howden W. E.* Symbolic Testing and the DISSECT Symbolic Evaluation System.— Computer Sci. Techn. Rep., Applied Physics and Information Science, University of California, San Diego. La Jolla, California, 1976, № 11.
9. *Balzer R. M.* EXDAMS – Extendable Debugging and Monitoring System.— In: Proc. 1969 Spring Joint Computer Conference. Montvale, New Jersey, 1969, p. 567–580.
10. *Howden W. E.* Symbolic Testing and the DISSECT Symbolic Evaluation System.— IEEE Trans. Software Eng., 1977, v. SE-3, № 4, p. 266–278.
11. *King J. C.* A New Approach to Program Testing.— In: Proc. 1975 Intern. Conf. on Reliable Software. Los Angeles, California, 1975, p. 228–233.
12. *King J. C.* Symbolic Execution and Program Testing.— Commun. ACM, 1976, v. 19, № 7, p. 385–394.
13. *Бичевский Я. Я.* Автоматическое построение полных систем примеров: Дис. на соискание уч. ст. канд. физ.-мат. наук. Киев: Ин-т кибернетики АН УССР, 1977.
14. *Бичевский Я. Я.* Автоматическое построение систем примеров.— Программирование, 1977, № 3, с. 60–70.
15. *Правильщиков П. А., Щепин В. С.* Составление структурных программ в диалоговом режиме с одновременной генерацией тестов.— Автоматика и телемеханика, 1979, № 8, с. 129–137.
16. *Позин Б. А.* Метод структурного построения тестов для отладки управляющих программ.— Программирование, 1980, № 2, с. 62–69.
17. *Барздинь Я. М., Бичевский Я. Я., Калниньш А. А.* Построение полной системы примеров для проверки корректности программ.— Уч. зап. Латв. гос. ун-та, Рига, 1974, т. 210, с. 152–187.
18. *Boyer R. S., Elspas B., Levitt K. N.* SELECT – A Formal System for Testing and Debugging Programs by Symbolic Execution.— In: Proc. 1975 Intern. Conf. on Reliable Software, Los Angeles, California, 1975, p. 234–245.
19. *Clarke L. A.* A System to Generate Test Data and Symbolically Execute Programs.— IEEE Trans. Software Eng., 1976, v. SE-2, № 3, p. 215–222.
20. *Howden W. E.* Methodology for the Generation of Program Test Data.— IEEE Trans. Computers, 1975, v. C-24, № 5, p. 554–560.

21. *Kandu S.* SETAR – A New Approach to Test Case Generation.– Software Testing, Infotech State of the Art Report, 1979, v. 2, p. 161–186.
22. *Борзов Ю. В.* Методы тестирования и отладки программ ЭВМ.– Рига: Латв. гос. ун-т, 1980.
23. *Osterweil L. J., Fosdick L. D.* DAVE – A Validation Error Detection and Documentation System for Fortran Programs.– Software – Practice and Experience, 1976, v. 6, p. 473–486.
24. *Касьянов В. Н.* Практический подход к оптимизации программ.– Новосибирск: ВЦ СО АН СССР, 1978.
25. *Бичевский Я. Я., Борзов Ю. В.* К вопросу о формальном описании функции, реализуемой программой.– В кн.: Тез. докл. Первой междунар. конф. молодых ученых «Проблемы проектирования и применения дискретных систем в управлении». Минск, 1977, с. 207–208.
26. *Майерс Г.* Надежность программного обеспечения. М.: Мир, 1980.

Поступила в редакцию
8.IV.1981

## AN EXTENSION OF METHODS FOR SYMBOLIC TESTING OF COMPUTER PROGRAMS

BICHEVSKIY Ya. Ya., BORZOV Yu. V.

The paper is concerned with application of symbolic testing of computer programs by automatic generation of input data (tests), complete systems of tests, identification of all realizable program paths, and generation of a description of the function to be implemented by the program. Various descriptions which completely specify the function to be implemented by the program are given.

## ЛИТЕРАТУРА


1. *Вишняков Ю. М., Кухарев Г. А.* Структура и основные принципы построения систем сбора и обработки экспериментальных данных в режиме разделения времени.— Респ. НТС «Опыт создания и внедрения автоматизированных систем обработки данных комплексных испытаний сложных объектов»: Тезисы докл. Киев: ИК АН УССР, 1978, с. 78—81.
2. *Оноэ М.* Метод двумерного преобразования данных без транспонирования большой матрицы данных.— ТИИЭР, 1975, № 1, с. 198—199.
3. *Кухарев Г. А.* Вычислительные процедуры быстрого преобразования Фурье для обобщенного спектрального анализа сигналов.— В сб.: Автоматизация проектирования систем автоматического управления. Л., Институт точн. мех. и опт., 1979, с. 72—78.
4. *Ахмед Н., Рао К., Шульц П.* Обобщенное дискретное преобразование.— ТИИЭР, 1971, № 9, с. 93—95.
5. *Rao K. et al.* Complex Haar Transform.— IEEE ASS P-24, 1976, № 1, p. 102—105.


Поступила в редакцию 13.VI.1980

УДК 681.3.06

# ПРИОРИТЕТЫ В ОТЛАДКЕ БОЛЬШИХ ПРОГРАММНЫХ СИСТЕМ

Бичёвский Я. Я., Борзов Ю. В.

**Рассматривается способ организации отладки больших программных систем при помощи макрокоманд, включаемых в разрабатываемые программы. Выполнение макрокоманд динамически управляется приоритетами, позволяющими без изменения разрабатываемых программ реализовать различные режимы отладки.**

Типичная схема отладки. Отладка программных систем обычно начинается с автономной отладки блоков. Вначале при помощи трансляторов выявляются и устраняются синтаксические ошибки, что, как правило, не составляет особого труда. Затем составляются тесты (входные данные), на которых проверяется работа программы. В большинстве случаев при этом обнаруживаются ошибки динамического характера. Нахождение их причин составляет, пожалуй, наиболее тонкую сторону работы программиста. Нередко оказывается что имеющейся информации недостаточно для определения причин ошибки. В таком случае для получения этой недостающей информации программист видоизменяет свою программу, чтобы обеспечить выдачу промежуточных результатов, содержимого различных переменных, историю передачи управления и т. п. Это достигается либо обращением к подпрограммам отладки, либо использованием отладочных средств языков (опция CHECK языка ПЛ/1), либо вставкой специальных отладочных макрокоманд, либо просто добавлением в программу дополнительных операторов. Наконец, дополнительную информацию можно получить, выполняя программу в интерпретирующем режиме. В любом случае отлаживаемая программа подобно строящемуся зданию «обрастает лесами». После окончания отладки программы эти «леса» убираются навсегда. Получение какой-либо дополнительной информации при аварии готовой программы обычно весьма затруднительно (приходится вновь «возводить леса»).

Следующая за автономной комплексная отладка — дело еще более сложное. Проверка работы программной системы при условии, что из автономных блоков отладочные средства не удалены, весьма проблематична из-за огромного объема выдач дополнительной информации и резкого замедления работы программы. Если же отладочные средства не использовать, то почти невозможно локализовать ошибку. Ввиду того, что стандартные средства отладки в основном предназначены для автономной отладки программ, они не дают возможность достаточно гибко и легко менять объем и уровень детализации получаемой отладочной информации.

Макрокоманды отладки и приоритеты. Авторы не намерены рассказывать о какой-нибудь новой системе отладки. Речь пойдет о принципе построения программной системы, при соблюдении которого с точки зрения тестирования разрабатываемая система будет иметь свойства, очень похожие на диагностические свойства электронной аппаратуры, а именно:

возможность снятия характеристик в заранее запланированных точках;

возможность возбуждения искусственного сигнала на заранее запланированных входах;

сохранение одних и тех же диагностических средств как во время разработки, так и во время эксплуатации.

Для обеспечения возможности снятия характеристик и возбуждения искусственного сигнала (в программировании это означает вывод содержимого областей памяти и ввод информации в определенные области памяти) следует иметь отладочные макрокоманды двух типов: макрокоманды вывода информации и макрокоманды ввода информации. Макрокоманды заранее вставляются в тело программы и там остаются навсегда.

Таким образом мы подготавливаем точки снятия характеристик и возбуждения сигнала. Понятно, что функционированием макрокоманд следует управлять извне. А именно, каждая макрокоманда содержит операнд, называемый *приоритетом макрокоманды*. Значение этого параметра задает программист при написании макрокоманды, и оно в дальнейшем не меняется. В начале выполнения каждой макрокоманды приоритет этой макрокоманды сравнивается с *текущим приоритетом системы*. Если приоритет макрокоманды ниже или равен текущему приоритету системы, то макрокоманда выполняется. В противном случае она не выполняется.

Текущий приоритет системы устанавливается следующим образом. В заказе на выполнение разрабатываемой системы каждому рабочему блоку присваивается определенный *приоритет блока*. Значениями приоритета блока являются пары чисел $(n_1, n_2)$, где

$$0 \leqslant n_1 \leqslant N, \quad 0 \leqslant n_2 \leqslant N,$$

а $N$ — фиксированное для данной системы число. Приоритет блока (0, 0) означает, что соответствующий рабочий блок в данном сеансе вообще не выполняется (даже не загружается в оперативную память и при работе системы он вообще может отсутствовать). Остальные приоритеты блока означают, что соответствующий рабочий блок должен быть выполнен и при его выполнении текущий приоритет системы устанавливается равный приоритету этого блока.

Предположим, что разрабатываемая система состоит из управляющего блока (УБ) и рабочих блоков (РБ). УБ является резидентом системы. В зависимости от заказа он загружает в оперативную память рабочие блоки и передает им управление. В рабочих блоках в виде макрокоманд вставлены обращения к программе, реализующей отладочные операции. Эту программу будем называть отладочным блоком (ОБ). Он физически включен в УБ и также хранится в оперативной памяти.

Если в данном сеансе некоторому рабочему блоку присвоен приоритет блока (4,3), то УБ в необходимый момент загружает в оперативную память этот РБ, устанавливает текущий приоритет системы, равный (4,3), и передает управление РБ. В течение работы блока будут выполнены все те отладочные макрокоманды ввода информации, которые имеют приоритет от 1 до 4, и все те отладочные макрокоманды вывода информации, которые имеют приоритет от 1 до 3.

Макрокоманды с приоритетами соответственно от 5 до $N$ и от 4 до $N$, хотя и они в программе могут присутствовать, выполнены не будут.

Ясно, что при разумном использовании приоритетов отладочных макрокоманд в зависимости от заказа на выполнение системы можно легко управлять тестированием и отладкой системы.

Дополнительные замечания. Макрокоманды отладки нет нужды менять, и они могут присутствовать в готовой системе. На быстродействие системы это будет влиять незначительно потому, что сравнение приоритета макрокоманды и текущего приоритета системы осуществимо несколькими машинными командами.

В частности, полезно оставлять в начале каждого РБ макрокоманды отладки, распечатывающие входную информацию, присвоив им низкий приоритет. Это *намного облегчит в будущем* локализацию ошибок.

Отлаженные РБ обычно выполняются с приоритетом 1, а макрокоманды с приоритетом 1 вообще не используются. Это позволяет выполнять отлаженные блоки без каких-либо отладочных действий.

Теперь рассмотрим имитацию работы несуществующих или неотлаженных РБ. Имитация реализуется включением макрокоманды отладки в начало следующего выполняемого РБ. Притом имитируемый РБ вообще не выполняется, т. е. ему присваивается нулевой приоритет блока.

Если при выполнении РБ приоритет этой макрокоманды ниже или равен текущему приоритету системы, то она создает информацию, которая соответствует результату работы предыдущего (но фактически невыполненного) РБ. Таким образом, получается, что каждый РБ может создавать данные для себя и отлаживаться автономно.

Если при выполнении РБ приоритет макрокоманды отладки выше текущего приоритета системы, то она не выполняется. Следовательно, РБ будет обрабатывать информацию, в действительности созданную предыдущим РБ.

Ясно, что этот метод имитации позволяет легко переходить от автономной отладки РБ к комплексной и обратно без всякого изменения разрабатываемой системы: меняется в заказе на работу системы только одна карта, задающая приоритеты выполнения РБ.

Опыт использования. Описанная схема с незначительными отличиями была использована при разработке системы СМОТЛ в 1974—1976 гг. сотрудниками ВЦ Латвийского госуниверситета. Система предназначалась для автоматического построения полных систем примеров. Ее объем порядка 30 000 операторов ЯСК, СМОД и специально разработанных макрокоманд обработки списков. В силу переборного характера реализованных алгоритмов система должна была быть максимально быстродействующей. Использовались отладочные макрокоманды распечатки списков и ввода списков. Отладочный блок занимал 2300 ячеек.

Разработка описанного выше отладочного блока потребовала примерно 4 человеко-месяца. Но его использование, как оказалось позже, сэкономило, по нашим расчетам, по крайней мере 3 человеко-года, так как разработка и отладка всех РБ проводилась параллельно. Комплексную отладку удалось начать уже через три месяца с начала разработки рабочих блоков. Кроме этого, один из РБ не удалось отладить в течение полугода после того, как все остальные РБ уже функционировали. Несмотря на это, схема отладки давала возможность имитировать работу этого РБ, а остальные блоки выполнять в реальном режиме. Оставленные в РБ макрокоманды отладки позволяют нам эффективно сопровождать систему.

В последующие годы описанная схема отладки была успешно применена еще в четырех разработках в ВЦ ЛГУ им. П. Стучки: создании одной СУБД, компилятора и интерпретатора ПЛ/1-программ, системы автоматизации подготовки инструментального производства.

Конкретный вид макрокоманд. Заметим, что нет практической возможности создания универсальных макрокоманд отладки, так как они сильно зависят от обрабатываемой информации. Следовательно, при создании новой программной системы почти всегда приходится заново создавать и макрокоманды отладки. Например, при разработке системы СМОТЛ, где основную форму обрабатываемой информации представляли списковые структуры, были запрограммированы следующие две макрокоманды отладки:

ВВЕСТИ (АДРЕС, ПРИОРИТЕТ),

где АДРЕС — переменная, содержащая адрес оперативной памяти, куда должна быть помещена вводимая списковая структура. ПРИОРИТЕТ — десятичная цифра от 1 до 7.

УДК 61[4.3.06

А. А. КАЛНИНЬШ. Ю. В. БОРЗОВ

# ТЕСТИРОВАНИЕ ПРОГРАММ: ЯЗЫКИ СПЕЦИФИКАЦИЙ И АВТОМАТИЧЕСКОЕ ГЕНЕРИРОВАНИЕ ТЕСТОВ[1]

## ВВЕДЕНИЕ

Тестирование как самостоятельная фаза разработки программ ЭВМ была, по-видимому, введена К. Бейнером в 1957 г. [1].

Программа сама по себе не является ни правильной, ни неправильной. Если она не содержит синтаксических ошибок и, следовательно, может быть запущена на ЭВМ, то обязательно вычисляет какую-то функцию (здесь «функция» понимается в самом широком смысле, как отображение входных данных в выходные). Правда, эта функция (функция программы) может значительно отличаться от той, для вычисления которой составлялась программа (желаемой функции). Цель процесса тестирования ограничивается констатацией ошибки (т. е. несовпадения функции программы и желаемой функции), если таковая имеется.

## 1. ЯЗЫКИ СПЕЦИФИКАЦИЙ

Очевидно, что тестирование ни отдельной программы, ни комплекса программ невозможно без точного знания их функционального назначения. Если средствам программирования, в особенности языкам программирования, уделяется большое внимание уже более двадцати лет, то разработка средств для описания объекта программирования началась примерно пять лет назад. Это так называемые языки спецификаций разного уровня. Конечно, в неявном виде они присутствовали с самого начала программирования как неформальные описания на естественном языке задания для программы. Сначала это были просто устные высказывания, а потом превратились в большие документы объемом в несколько сот страниц и более. Но основные их недостатки как для программирования, так и для тестирования остались. Это частые ошибки, неточность, двусмысленность, неполнота, противоречивость различных частей. В последнее время делаются попытки уточнить с помощью строгой методологии спецификации на естественном языке [2, 3], но для автоматизации процесса производства программ это пока не дало результатов. В последние годы быстро начали развиваться полуформальные и формальные языки спецификаций. По своему назначению они бывают двух типов — для общего описания требований на систему программ в целом, т. е. для формального описания так называемого технического задания на систему и для описания конкретного функционирования системы, как правило, уже на уровне отдельных подсистем и ее программ, т. е. на уровне технического проекта. Языки первого типа будем называть языками описания требований (requirements), а языки второго — языками описания проектов (design).

### 1.1. Языки описания требований

Эти языки служат для описания общей структуры системы, объектов внешнего мира, взаимодействующих с системой, типов их воздействия и реакцией системы на эти воздействия. Разработка систем, как правило, начинается с составления спецификации на таком уровне.

В настоящее время известно несколько языков подобного типа, отличающихся степенью формализованности: RSL [4], PRL [5], PSL [6], HIPO [7, 8], AXES [9], AKT [10], язык описания программ в системе ЯУЗА-6 [11]. Все они в некоторой степени являются попытками формализовать и объединить понятие блок-схемы для изображения течения управления в системе и понятие схемы потоков данных. В некоторых из них вводится еще понятие состояния системы, но все эти языки характеризуются тем, что объекты, условия, производимые действия и выдаваемые результаты (реакции системы) не конкретизируются, а остаются на уровне неинтерпретированных символьных названий, поэтому они не функциональны, т. е. невозможно формально по конкретным входным данным или воздействиям узнать из спецификации, каковы должны быть результаты (реакции) системы. Языки RSL, PRL, AXES и язык системы ЯУЗА-6 ориентированы на спецификацию систем реального времени, PSL — на системы обработки информации, HIPO, AKT — на различные применения. Более формализованными из них являются RSL и PRL. Для всех этих языков характерна иерархичность — действие одного уровня можно дальше уточнить более подробной схемой на другом уровне.

Приведем отрывок из примера [5], описывающего действия карманного калькулятора на языке PRL:

Внешние объекты: ПОЛЬЗОВАТЕЛЬ, РЕГИСТР_X, РЕГИСТР_Y....

Входные воздействия: ВВОДИТ_ЦИФРУ, ВВОДИТ_., ВВОДИТ_=,...

[1] Данная статья—сокращенный вариант доклада «Инвентаризация идей тестирования программ», прочитанного авторами на Всесоюз. науч. конф. «Синтез, тестирование, верификация и отладка программ» 22—24 сентября 1981 г. в Риге.

```
Реакция системы: ПЕРЕСЛАТЬ_ЦИФРУ_В, ОЧИСТИТЬ,
                 ПОКАЗАТЬ_ НА ТАБЛО, ПЕРЕСЛАТЬ,...
Состояния: ГОТОВ_К_ВВОДУ_ЧИСЛА,
    ГОТОВ_К_ВВОДУ_ЦИФРЫ,...
TITLE: КАЛЬКУЛЯТОР
FEATURE: ФУНКЦИЯ_ВВОДА_ЧИСЛА
ГОТОВ_К_ВВОДУ_ЧИСЛА/* состояние, где применима
    функция*/
IF ПОЛЬЗОВАТЕЛЬ_ВВОДИТ_ЦИФРУ OR
    ПОЛЬЗОВАТЕЛЬ_ВВОДИТ_.
THEN ПЕРЕСЛАТЬ   РЕГИСТР_X В РЕГИСТР_Y/*
        1-я реакция*/
        ОЧИСТИТЬ РЕГИСТР_X/* следующая реакция*
        РЕГИСТР_X ПОКАЗАТЬ_НА_ТАБЛО
IF ПОЛЬЗОВАТЕЛЬ ВВОДИТ_ ЦИФРУ * в IF используется тип воздействия*/
THEN ПЕРЕСЛАТЬ_ ЦИФРУ_ В РЕГИСТР _X
ELSE ...
```

В этом примере все слова на английском языке — ключевые слова PRL, имеющие стандартную семантику, как и в языках программирования, /* ограничивает комментарии.

Запись этого же фрагмента на языке RSL [4] выглядела бы похоже, но только в языке PRL имеется возможность более строгого описания объектов (называемых элементами и данными) и действий (называемых альфами) и, кроме того, более сложных структур управления (например, параллельное ветвление), описываемых на языке $R$-сетей.

Языки спецификации этого вида служат в основном для уточнения технического задания, исключения ошибок в нем, улучшения его однозначности. Некоторые свойства таких спецификаций можно проверить автоматически, например, задана ли для всех состояний и всех возможных в них входных воздействий одна и только одна реакция системы. Поэтому эти языки связаны с соответствующими технологическими комплексами, которые транслируют их, записывают результат трансляции в соответствующую базу данных, проверяют полноту и непротиворечивость задания, готовят различного рода документы (для языка RSL это система REVS [4], для языка описаний программ — ЯУЗА-6 [11] и т. д.).

Однако языки такого типа имеют непосредственное применение в тестировании: они дают возможность формализовать и частично автоматизировать комплексное тестирование всей системы. Так как в конце разработки все объекты, данные, действия и реакции имеют однозначную реализацию (программную или аппаратную), то спецификация становится выполнимой и возможна проверка соответствия системы.

### 1.2. Языки описания проектов

Языки спецификации второго типа имеют для тестирования значительно большее значение. Они предназначены для полного функционального описания работы системы программ — при каких входных данных (воздействиях) какие выходные данные (реакции) должны появляться. Эти языки должны описывать соотношение вход-выход, но необязательно — описывать алгоритм, с помощью которого выход получается. Они в значительной степени напоминают языки программирования очень высокого уровня. Главное в них — простота и удобство записи, а оптимизация и удобство реализации совсем игнорируется. Требования к таким языкам исследованы в [12]. Для многих из них можно создать трансляторы на обычные языки программирования, но программы получаются настолько неоптимальными, что пригодны только для целей тестирования. Задача оптимизации для них фактически должна включить весь существующий опыт программирования и поэтому пока вряд ли автоматизируема. Существенным моментом здесь может оказаться также то, что спецификация отражает только некоторые аспекты задачи и в этом смысле является неполной.

Пока очень мало языков, созданных специально для спецификации программ, ни один из них еще не обладает всеми желаемыми свойствами, однако в некоторых имеющихся языках содержатся идеи, которые перспективны для использования в языках спецификаций.

Во-первых, это идея использования известных больших операций над математическими объектами. Примером может служить язык СЕТ.Т [13], где коротко, не заботясь о реализации, можно записать действия над множествами, но в нем нет средств расширения, позволяющих ввести такие действия с другими объектами.

Далее, это язык исчисления предикатов, дополненный конкретными функциями и предикатами для определенного класса задач. Заметим, что такой язык спецификаций используется в большинстве случаев для доказательства правильности программ [14, 15]. Основная цель введения здесь дополнительных предикатов — это возможность записи формулы без кванторов. Например, для класса задач сортировки [15] обычно используются предикаты ORDERED, PERM и функции выбора различных подмассивов. В общем случае языков спецификаций бескванторная запись необязательна, конкретные предикаты вводятся для выделения базовых понятий класса задач.

Перспективен подход, связанный с абстрактными типами данных (или идей кластеров). Абстрактный тип данных — это множество объектов с набором атрибутов и определенными на них операциями. Абстрактные типы данных, как правило, образуют уровни иерархии. Этот механизм реализован в некоторых языках программирования, например Alphard [16]. Основная проблема — это определение семантики вводимых операций. Средства, применяемые в языках программирования (определение через реализацию с помощью базовых операций и условия верификации), в случае языков спецификаций неудобны. Более подходя-

шим является определение семантики вводимых операций с помощью алгебраических тождеств [17], например POP (PUSH (A, X)) = A, где A — стек. Но трудной является проблема подбора полного и удобного для работы набора тождеств.

Плодотворны идеи перехода на функциональный стиль записи действий над информацией [18] и поуровневое описание действий. С помощью этих и, может быть, еще других идей возможно образование некоторого ядра языков спецификаций. Практическое применение, вероятно, получат проблемно-ориентированные языки спецификаций — расширение этого ядра наиболее типичными понятиями, сложными типами данных и «большими» операциями для данной проблемной области.

Наиболее исследованная область к настоящему моменту — это задачи обработки экономической информации. Языки спецификаций здесь могут быть двух уровней — для описаний отдельных программ с учетом конкретного представления входных и выходных данных и для описания группы программ на уровне реализуемого ею абстрактного алгоритма, как правило, без уточнения представления информации.

Несколько простых языков первого уровня для задач последовательной обработки файлов разработаны в ВЦ ЛГУ им. П. Стучки. Два из них [19, 20] — предикатного типа. Язык из [19] предназначен для описания простых неарифметических преобразований файла и допускает запись типа $\forall x\, R(x, A, B)$, где $x$ — запись, $A$ — входной файл, $B$ — выходной файл, $R$ — предикат, построенный из отношений между полями записей и некоторых других отношений, например $x \in A$. Практическое применение может иметь язык L1 [20], допускающий более сложные кванторные префиксы и выражения. В нем явно введено понятие подфайла — группы рядом стоящих записей с данным свойством, определяемой регулярной формулой (например, группа, у которой одно из полей постоянно), и понятие вертикальной операции, например суммы некоторого поля по всем записям данного подфайла. На понятиях подфайла и действий с ним базируется также язык спецификаций BR [21], однако в нем используется функциональная запись преобразований файла. Языки спецификаций этого уровня могут служить для тестирования отдельных программ обработки файлов.

Языки спецификаций второго уровня для задач обработки данных перспективнее для тестирования, но пока менее развиты. Дополнительной проблемно-ориентированной идеей здесь является использование реляционной базы данных [22] как основной модели представления информации. Первым из языков такого типа является язык BDL [23], но он по форме больше напоминает язык программирования. Язык, описанный в [24], ориентирован на спецификации и использует обобщение понятия реляционной базы данных для представления обрабатываемой информации (база может быть иерархической — элементом отношения снова может быть отношение) и функциональную запись. Введен набор больших операций для глобального преобразования данных, например добавление новых элементов в отношении, их изменение, перегруппировка и др.

Явно на спецификацию ориентирован язык HISP [25], базирующийся на иерархическом применении кластера — группы связанных типов данных с операциями над ними. Связи между операциями задаются с помощью алгебраических тождеств.

В некоторой степени языком спецификаций является также *P*-метаязык в технологическом комплексе РТК [26].

По уровню и принципам построения близки к языкам спецификаций и некоторые языки, используемые в задачах искусственного интеллекта. Среди них выделяется язык PLASMA [27], специально ориентированный на процедурные спецификации программ. Главная его ценность — развитые средства сравнения по образцу, позволяющие, например, разбить строку на подстроки указанного вида. Интересны также языки, использующие понятие фрейма, которое обобщает как реляционные базы данных, так и абстрактные типы. От них можно заимствовать, например, понятие асинхронных процедур, связанных с некоторым типом данных и выполняющихся при любых операциях с этими данными.

Появление языков спецификации проектов, вероятно, строго изменит состояние тестирования программ — даст новые критерии полноты тестирования, алгоритмы построения тестов. Однако они вызовут и новую проблему — тестирование спецификации, которое, как правило, остается на неформальном уровне. Но тестирование их может оказаться более легким процессом, так как языки спецификаций более ориентированы на анализ и понимание, чем языки программирования.

### 1.3. Языки тестирования

Некоторой разновидностью языков спецификаций являются языки тестирования, которые служат для описания самого процесса тестирования.

В простейшем случае язык тестирования дает возможность записать один или несколько комплектов тестовых данных и соответствующих им предполагаемых результатов работы тестируемой программы. Приведем пример фрагмента такой записи на языке тестирования TPL/F [28, 29]:

```
SUB1 : K = 2
SUB1 : N = 3
VERIFY (SUB1 : K. EQ. 8. AND. SUB1 : N. EQ. 9)
EXECUTE
```

Первые две строки означают, что переменные K и N тестируемой подпрограммы SUB1 должны вначале содержать 2 и 3. Третья строка задает предполагаемый результат — те же переменные должны быть равными 8 и 9. Последняя строка является командой выполнения. Видно, что тестовая процедура записана на уровне входного языка программиро-

вания—ФОРТРАН. Подобный язык для КОБОЛа описан в [30].

Язык тестирования является входным языком для технологической системы — тестового выполнителя [29], который создает среду для выполнения отдельных модулей, устанавливает начальные значения, имитирует выполнение тестируемого модуля, и несуществующих модулей, а также проверяет полученные результаты.

Дальнейшее развитие языков тестирования связано с языками спецификации проектов. Почти из каждого такого языка можно получить язык тестирования, если к нему добавить сравнительно несложные средства для описания подготовки тестов и запуска программ. Языки тестирования такого типа позволяют автоматически проверить результат выполнения не только на нескольких заданных примерах, но и на произвольном примере, поставляемом, например, некоторой процедурой автоматической генерации тестов. Несуществующие модули при этом можно заменить их описаниями на языке спецификаций. Одним из первых языков тестирования такого типа является язык FTL [31], построенный на базе идей языка спецификаций BR[21] и языка программирования ПЛ—1. Вообще, использование в качестве языков тестирования, возможно, будет главным практическим применением для многих языков спецификаций.

## 2. ГЕНЕРИРОВАНИЕ ТЕСТОВ

### 2.1. Теоретические вопросы генерирования тестов

После того как зафиксирован критерий выбора тестов, начинается само генерирование тестов. Из результатов теории алгоритмов следует, что для реально используемых критериев не существует алгоритма, автоматически строящего набор тестов согласно этому критерию для произвольной программы. Все положительные результаты получены для частных классов программ.

Критерием выбора тестов, как правило, является $C_1$ — прохождение всех ветвей. Обычно строится некоторая система путей, содержащая все ветви программы. Часть работ [32, 33] связана с графотеоретическим аспектом нахождения такого покрытия ветвей путями. Основные же трудности связаны с нахождением реализуемых путей, покрывающих все ветви, и с самим нахождением тестов, соответствующих данному пути. Обе эти проблемы фактически во всех случаях решаются с помощью символического выполнения программ [34—39], которое для данного пути дает условие его реализуемости — систему равенств и неравенств над входными данными программы (входными параметрами, файлами и т. д.), при выполнении которой проходится данный путь. Если условие реализуемости не имеет решения, то путь не реализуем. В общем случае нет алгоритма, проверяющего реализуемость пути, но для некоторых классов программ [35, 36, 38, 39] можно ограничиться линейными равенствами и неравенствами, для решения которых используются как стандартные [35, 38, 39], так и специальные алгоритмы [36].

Дальнейшая проблема связана с достижением данной ветви программы реализуемыми путями. Она является алгоритмически разрешимой только для некоторых классов программ, а именно для так называемого базового языка [40, 41], предназначенного для описания преобразований файлов, где единственный тип данных — целые числа, и допустимые операции — чтение числа с некоторой входной ленты (входного файла), пересылка между внутренними переменными, арифметическое сравнение переменных между собой и с константами, запись на выходную ленту (программа образуется обычным образом с помощью операторов IF и GO TO). Фактически этот язык определяет класс многоленточных автоматов с бесконечным алфавитом. Некоторые расширения базового языка сохраняют разрешимость, например ограниченное введение цикла типа FOR [40], одного счетчика [42] магазина [43], однако дальнейшие допущения арифметики делают проблему достижимости неразрешимой [42, 44]. Следовательно, она неразрешима для всех обычных языков программирования, поэтому для них, как правило, используются эвристические, неполные алгоритмы. Все положительные результаты в проблеме достижимости получены с помощью понятия состояния [40, 41, 45].

Состояние программы после прохождения данного пути — это объект, получаемый из условия реализуемости данного пути и содержащий те соотношения между переменными и другими объектами программы, которые достаточны для выяснения реализуемости любого продолжения данного пути. Для разных классов программ используются разные конкретные определения состояния, и лишь для классов, где состояние определено так, что для каждой программы существует только конечное число различных состояний, удается доказать разрешимость проблемы достижимости. Например, для базового языка [40, 41] состояние — это набор тех равенств и неравенств между внутренними переменными программы, которые выполняются после прохождения данного пути, и таких наборов заведомо может быть только конечное число.

Понятие состояния оказалось очень полезным для других целей анализа программы, особенно при верификации программ с помощью тестов.

Проблемы генерации тестов для языков спецификаций, по-видимому, аналогичны проблемам для языков программирования, но к настоящему моменту они мало исследованы.

### 2.2. Практика автоматического генерирования тестов

Первая автоматическая система генерирования тестов была создана более 20 лет тому [46]. Суть метода состояла в добавлении к тестируемой КОБОЛ-

программе специальной секции, описывающей отношения между данными. Исходя из этой секции, заданной пользователем, с помощью псевдослучайных чисел генерировались тесты.

Следующая известная система была создана почти через 10 лет [47], и только в первой половине 70-х годов появилось семейство полуавтоматических систем, автоматизировавших отдельные фазы генерирования тестов:

а) выбор множества путей, в совокупности покрывающих все дуги управляющего графа (т. е. все ветви) программы [48];

б) генерирование по данному пути программы условия его реализуемости [49].

Последние дали начало системам тестирования программ, основанным на символическом выполнении программы [34]. Во второй половине 70-х годов был создан ряд экспериментальных систем автоматического генерирования тестов с применением символического выполнения [35—39] и предложена группа методов [50—52], являющихся модификациями символического выполнения.

Система СМОТЛ выделяется среди вышеперечисленных главным образом тем, что она основывается на формальной математической теории [41]. Понятие состояния, играющее ключевую роль в теории автоматического посторения полных систем тестов, здесь действительно было применено в системе автоматического генерирования тестов, что дает гарантию построения набора тестов, удовлетворяющего критерию $C_1$ для любой программы. Точнее, такой набор тестов гарантируется для программ, где не происходит деформации значений переменных, определяющих передачи управления, т. е. значения не входят в сложные арифметические выражения, не сдвигаются и т. п. Экспериментальная эксплуатация системы на реальных программах показала ее жизнеспособность. Система СМОТЛ действительно смогла построить наборы тестов, удовлетворяющие критерию $C_1$, для большинства реальных программ обработки данных из нескольких сот операторов за время, приблизительно равное времени трансляции этих программ. Однако небольшое распространение входного языка системы (СМОД-язык обработки данных, родственный КОБОЛу), прекращение выпуска ЭВМ Минск-32, а также отсутствие возможностей автоматически проверить результаты тестирования препятствовали широкому внедрению данной системы. По мнению авторов, основной предпосылкой для практического применения автоматизации генерирования тестов данного типа является разработка больших технологических комплексов, включающих как составную часть не только генерацию тестов, но и автоматизированную проверку результатов тестирования с помощью формализованных спецификаций.

Другим важным (и исторически первым) направлением в автоматическом генерировании тестов следует назвать подход, основанный на применении псевдослучайных чисел [53], так называемое стохастическое тестирование, которое особенно себя зарекомендовало при тестировании систем типа трансляторов, где оно используется в сочетании с аппаратом грамматики, описывающим входные последовательности символов [26].

Интересно отметить наблюдаемый перенос методов технической диагностики на программы [33, 39, 54], а в последнее время также методов тестирования программ на тестирование электронной аппаратуры [55]. Безусловно перспективен подход, предполагающий выделение определенных типов ошибок и предусматривающий методы выявления ошибок для каждого их типа [56].

## 3. АНАЛИЗ ПРОГРАММ С ПОМОЩЬЮ ТЕСТИРОВАНИЯ

### 3.1. Генерирование функции программы

Одним из возможных подходов к тестированию программ является формальное описание отображения, осуществляемого программой (т. е. в некотором смысле восстанавливается спецификация по программе), которое затем используется для выяснения правильности преобразования, фактически выполняемого программой. Например, оно может сравниваться с данной спецификацией задачи или проверяться пользователем.

В случае обработки данных генерирование функции программы означает построение для программы множества входных файлов и отображение входных файлов на выходные. В [57] вводится язык описания таких отображений и доказывается, что для базового языка, являющегося упрощенной моделью языков программирования типа обработки данных, справедливо следующее: существует алгоритм А, который для каждой программы П на базовом языке генерирует конечную систему формул обработки файлов А (П), описывающую то же преобразование, что и программа П.

Аналогичные исследования для языка символьного процессора рассматриваются в [58].

Следует отметить, что практическая ценность данных методов фактически зависит от наглядности, легкости восприятия генерируемых описаний отображений — функций программы. Идеально было бы получить описание на каком-то из удобных языков спецификаций. В силу алгоритмических трудностей кажется реальным получить описание функции программы только на языках спецификации низкого уровня, примером которых может служить и язык из [57].

### 3.2. Тестирование в качестве верификации

Тестирование в обычном смысле может только показать наличие ошибок, но не их отсутствие, однако многие идеи и приемы тестирования могут быть использованы для верификации программ.

Под верификацией будем понимать точную проверку соответствия программы ее спецификации.

Для некоторых классов языков программирования и языков спецификаций существуют алгоритмы верификации, которые фактически определяют конечный полный набор тестов. В качестве простого примера заметим, что алгоритм проверки соответствия диаграммы конечного автомата его спецификации (регулярной формуле или формуле метаязыка И [59]) фактически определяет конечное множество тестов — слов данной длины, на которых и проверяется соответствие.

Для базового языка программирования имеется несколько языков спецификаций — кванторный язык [19], функциональный язык BR [21] и язык формул обработки файлов [57], для которых существует алгоритм верификации. Во всех этих случаях этот алгоритм можно интерпретировать так, что спецификация сначала преобразуется в программу, которая символически выполняется совместно с данной программой на конечном множестве путей. При совпадении символических результатов и получается соответствие программы спецификации. Множество путей определяется так, чтобы были достигнуты все определенные состояния при параллельном выполнении обеих программ. Аналогичная техника применяется при верификации программ языка ЛЕМ [60], в котором значительно усложняется определение символического выполнения и понятие состояний, с соответствующим языком спецификаций [20, 61] кванторного типа.

В [19] приводится одна принципиально иная идея верификации с помощью тестирования для базового языка. Показано, как верификацию можно свести к проблеме достижимости, а именно: как для данной спецификации надо преобразовать каждую команду в программе в некоторый блок таким образом, что преобразованная программа достигает некоторую команду тогда и только тогда, когда исходная программа не соответствует спецификации. Затем из разрешимости проблемы достижимости следует разрешимость верификации.

Для более богатых языков не могут существовать полные алгоритмы верификации, но идеи тестирования используемы и при практических, эвристических приемах верификации в этих языках.

## ЗАКЛЮЧЕНИЕ

Тестирование программ особенно бурно развивалось в 1973—1977 годах. В последние годы появилось немного новых идей, а в некоторых областях, как, например, в автоматическом генерировании тестов, не было заметного продвижения. Наверное, самая плодотворная идея из недавно появившихся — это формализованные языки спецификаций для описания проектов. На ее базе ожидаются новые результаты как в автоматизации генерирования надежных наборов тестов, так и в оценке результатов тестирования. В практическом плане самым важным было появление технологических комплексов программирования, объединяющих все разрозненные идеи тестирования. Именно они могут привести к массовому внедрению новых идей и принципов тестирования.

## СПИСОК ЛИТЕРАТУРЫ


1. Пархоменко П. П., Правильщиков П. А. Диагностирование программного обеспечения: Обзор.— Автоматика и телемеханика, 1980, № 1, с. 103—121.
2. Ross D., Shoman K. Structured analysis for requirements definition.— IEEE Trans. Software Eng., 1977, SE—3, N 1, p. 6—15.
3. Balzer R., Goldman N., Wile D. Informality in program specifications.— Ibid, 1978, SE—4; N 2, p. 94—103.
4. Bill T. E., Bixler D. C., Dyer M. E. An extendable approach to computer—aided software requirements engineering.— Ibid., 1977, SE—3, N 1, p. 49—60.
5. Davis A. M. Formal techniques and automatic processing to ensure correctness in requirements specifications.— In: Proc. IEEE conf. spec. rel. software, 1979, p. 15—35.
6. Teichroew D., Hershey E. A. PSL/PSA: A computer aided technique for structured documentation and analysis of information processing systems.— IEEE Software Eng., 1977, SE—3, N 1, p. 41—48.
7. Jones M. N. HIPO for developing specifications.— Datamation, 1976, N 3, p. 112—125.
8. Майерс Г. Надежность программного обеспечения.— М. : Мир, 1980.— 360 с.
9. Hamilton M., Zeldin S. Higher order software—a methodology for defining software.— IEEE Trans. Software Eng., 1976, SE—2, N 2, p. 9—32.
10. Фуксман А. В. Технологические аспекты создания программных систем.— М. : Статистика, 1979.— 184 с.
11. Липаев В. В. Проектирование математического обеспечения АСУ.— М. : Сов. радио, 1977.— 400 с.
12. Balzer R., Goldman N. Principles of good software specification and their implications for specification languages.— In: Proc. IEEE conf. specif. rel. software, 1979. New York: IEEE, 1979, p. 58—67.
13. Черноброд Л. В. Обзор основных конструкций языка программирования СЕТЛ.— В кн.: Системное и теоретическое программирование. Новосибирск: ВЦ Сиб. отд-ния АН, 1972, с. 236—277.
14. Luckham D. C. Program verification and verification oriented programming.— In: Information processing 77: Proc. IFIP congr. 77. New York: North-Holland, 1977, p. 783—793.
15. Непомнящий В. А., Чурина Т. Г. Верификация программ сортировки массивов.— В кн.: Языки и системы программирования. Новосибирск: ВЦ Сиб. отд-ния АН, 1979, с. 21—36.
16. Wulf W. A., London R., Shaw M. An introduction to the construction and verification of Alphard programs.— IEEE Trans. Software Eng., 1976, SE—2, N 4, p. 253—265.
17. Guttag J. V., Horowitz E., Musser D. R. Abstract data types and software validation.— Comm. ACM, 1978, 20, N 12, p. 1048—1064.
18. Backus J. Can programming be liberated from von Neuman style? A functional style and its algebra of programs.— Comm. ACM, 1978, 21, N 8, p. 613—641.
19. Barzdin J. M. The problem of reachability and verification of programs.— Lect. Notes Comput. Sci., 1979, 74, p. 13—25.
20. Августон М. И. Язык спецификации программ обработки данных.— В кн.: Синтез, тестирование, верификация и отладка программ: Тез. докл. Всесоюз. конф. Рига: изд-во Латв. ун-та, 1981, с. 18—19.
21. Калнинь Ш. А. Функциональные языки и корректность программ обработки файлов.— В кн.: Методы математической логики в проблемах искусственного интел-

лекта и систематическое программирование: Тез. докл. Всесоюз. конф. Вильнюс: Ин-т математики и кибернетики АН ЛитССР, 1980. с. 106—107.
22. Codd E. F. A relational model of data for large shared data banks.— Comm. ACM, 1970, 13, N 6, p. 377—388.
23. Avery high level programming language for data processing applications / M. Hammer, W. G. Howe, W. J. Kruskal, I. Wladowsky.— Ibid., 1977, 20, N 11, p. 832—840.
24. Зариньш А. К., Каксис Г. Язык спецификации задач обработки данных.— В кн.: Синтез, тестирование, верификация и отладка программ: Тез. докл. Всесоюз. конф. Рига: Изд-во Латв. ун-та, 1981, с. 105—106.
25. Futasugi K., Okada K. Specification writing as construction of hierarchically structured clusters of operators.— In: Information processing 80: Proc. IFIP congr. 80 New York: North-Holland, 1980, p. 287—292.
26. Вельбицкий И. В., Ходаковский В. Н., Шолмов Л. И. Технологический комплекс производства программ на машинах ЕС ЭВМ и БЭСМ-6.— М. : Статистика, 1980.— 263 с.
27. Hewitt C. E., Smith B. Towards a programming apprentice.— IEEE Trans. Software Eng., 1975, SE—1, N 1, p. 26—45.
28. Panzl D. J. Test procedures: A new approach to software verification.— In: Proc. 2nd Intern. conf. software eng., 1976. New York, 1976, p. 477—485.
29. Panzl D. J. Automatic software drivers.— Computer, 1978, 11, N 4, p. 44—50.
30. Doleman P. D. The Testscript language.— Test. Techn. Newsletter, 1979, 2, p. 3.
31. Калиниш А. А., Стродс Ю. Ф. Средства тестирования программ последовательной обработки файлов.— В кн.: Синтез, тестирование, верификация и отладка программ: Тез. докл. Всесоюз. конф. Рига: Изд-во Латв ун-та, 1981, с. 116—117.
32. Gabow H. N., Maheshwari S. N., Osterweil L. J. On two problems in the generation of program test paths.— IEEE Trans. Software Eng., 1976, SE-2, N 3, p. 227—231.
33. Евстигнеев В. А. Теоретико-графовый подход к верификации и тестированию программ.— В кн.: Синтез, тестирование, верификация и отладка программ: Тез. докл. Всесоюз. конф. Рига: Изд-во Латв. гос. ун-та, 1981, с. 92—93.
34. Ramamoorthy C. V., Ho S.— B. F., Chen W. T. On the automated generation of program test data.— IEEE Trans. Software Eng., 1976, SE — 2, N 4, p. 293—300.
35. SMOTL — A system to construct samples for data processing program debugging / J. Bičevskis, J. Borzovs, U. Straujums, A. Zarins, E. F. Miller, Jr.— Ibid., 1979, SE—5, N 1, p. 60—66.
36. Boyer R. S., Elspas B., Levitt K. N. SELECT — a formal system for testing and debugging programs by symbolic execution.— In: Proc. Intern. conf. rel. software, 1975, Los Angeles, 1975, p. 234 — 245
37. Clarke L. A. A system to generate test data and symbolically execute programs.— IEEE Trans. Software Eng. 1976, SE—2, N 3, p. 215—222.
38. Правильщиков П. А. Построение тестов для программ.— Автоматика и телемеханика, 1977, № 5, с. 147—160.
39. Barzdin J. M., Bicevskis J. J., Kalninsh A. A. Automatic construction of complete sample system for program testing.— In: Information processing 77: Proc. IFIP congr. 77. New York: North-Holland, 1977, p. 57—61.
41. Барздинь Я. М., Бичевский Я. Я., Калиниьш А. А. Построение полной системы примеров для проверки корректности программ.—В кн.: Теория алгоритмов и программ. Рига: Изд-во Латв. ун-та, вып. 1, с. 152—187.
42. Калиниьш А. А., Бичевский Я. Я., Барздинь Я. М. Разрешимые и неразрешимые случаи проблемы построения полной системы примеров.— Там же, с. 188—205.
43. Аузиньш А., Калис А. О разрешимости проблемы построения полной системы примеров для программ с магазинной памятью.— В кн.: Синтез, тестирование, верификация и отладка программ: Тез. докл. Всесоюз. конф. Рига: Изд-во Латв. ун-та, 1981, с. 20.
44. Барздинь Я. М., Калиниьш А. А. Построение полных систем примеров для программ, работающих с прямым методом доступа.— В кн.: Теория алгоритмов и программ. Рига: Изд-во Латв. ун-та, 1975, вып. 2, с. 123—154.
45. Бичевский Я. Я. Автоматическое построение систем примеров.— Программирование, 1977, № 3, с. 60—70.
46. Sauder R. L. General test data generator for Cobol.— In: AFIPS conf. proc., SJCC, 1962. AFIPS Press, 1962, p. 317—323.
47. Hanford K. V. Automatic generation of test cases.— IBM Systems J., 1970, 9, N 4, p. 242—257.
48. Miller E. F., Melton R. A. Automatic generation of testcase datasets.— In: Proc. Intern. conf. rel. software, 1975. Los Angeles, 1975, p. 51—58.
49. Balzer R. M. EXDAMS — Extendable debugging and monitoring system.— In: 1969 SJCC, AFIPS conf. proc., 1969, p. 567—580.
50. Huang J. C. An approach to program testing. — Comput. Surveys, 1975, 7, N 3, p. 113—128.
51. Howden W. E. Methodology for the generation of program test data.— IEEE Trans. Comput., 1975, C—24, N 5, p. 554—560.
52. Kundu S. SETAR — A new approach to test case generation.— In: INFOTECH state of the art report: Software Test., 1979, 2, p. 161—186.
53. Moranda P. B. Asymptotic limits to program testing.— Ibid., p. 201—212.
54. Данилов В. В., Соловей Г. Б., Филлипов В. Ф. Построение теста поиска дефектов для программ.— В кн.: Синтез, тестирование, верификация и отладка программ: Тез. докл. Всесоюз. конф. Рига: Изд-во Латв. ун-та, 1981, с. 84.
55. Тадевосян А. Г. О разрешимости проблемы построения полной системы примеров для одного класса программ.— Докл. АН АрмССР, 1981, 22, № 3, с. 151—156.
56. Архангельский Б. В., Никитин А. И., Черняховский В. В. Устойчивые типы семантических ошибок программ.— В кн.: Синтез, тестирование, верификация и отладка программ: Тез. докл. Всесоюз. конф. Рига: Изд-во Латв. гос. ун-та, 1981, с. 17.
57. Борзов Ю. В. Автоматическое построение описания функции, реализуемой программой ЭВМ.— Рукопись деп. в ВИНИТИ 81.02.25, № 887—81 Деп.
58. Ефимкин К. Н., Задыхайло И. Б. О верификации программ на одном языке.— Программирование, 1980, № 2, с. 69—77.
59. Трахтенброт Б. А., Барздинь Я. М. Конечные автоматы: Поведение и синтез.— М. : Наука, 1970.— 400 с.
60. Аугустон М. И. ЛЕМ — средство программирования обработки данных.— Программирование, 1978, № 1, с. 31—38.
61. Augustson M. I. Writing and verifying sequential files updating programs.—Lect. Notes Comput. Sci., 1978, 64, p. 102—111.

*Поступила 05.01.82*

Списковая структура кодируется и вводится с перфокарт.

ВЫВЕСТИ (АДРЕС, КОЛИЧЕСТВО_УРОВНЕЙ, ПРИОРИТЕТ), АДРЕС — переменная, содержащая адрес оперативной памяти, откуда должен быть распечатан список; КОЛИЧЕСТВО_УРОВНЕЙ — количество распечатываемых уровней списковой структуры; ПРИОРИТЕТ — десятичная цифра от 1 до 7.

* * *

Описанная схема проектирования большой программной системы характеризуется полным интегрированием средств отладки в саму разрабатываемую систему. Средства отладки таким образом становятся неотделимой частью системы, функционирующей все время жизни системы. Использование механизма приоритетов и возможность гибко менять режим отладочных действий на уровне языка управления заданиями делают такой подход особенно эффективным. При этом существенно облегчается сопровождение разработанной системы.

Поступила в редакцию 23.IX.1980
Переработанный вариант 26.XI.1981

УДК 681.3.06

# ПРИНЦИПЫ ПЛАНИРОВАНИЯ РЕШЕНИЯ ЗАДАЧ В СИСТЕМЕ АВТОМАТИЧЕСКОГО СИНТЕЗА ПРОГРАММ

Лавров С. С., Залогова Л. А., Петрушина Т. П.

Рассматриваются два метода перевода описаний моделей предметных областей и постановок задач с языка «Декарт» на язык логики предикатов первого порядка. Излагается метод доказательства существования решения задачи и извлечения программы из этого доказательства.

Большинство современных пакетов прикладных программ (ППП) представляют собой сложные программные комплексы, включающие специальный входной язык для описания задач и управляющую программу, которая по формулировке задачи на языке пакета планирует процесс вычислений и генерирует программу, реализующую поставленную задачу. Таким образом, при автоматическом способе организации вычислительного процесса пользователю достаточно указать только исходные данные и конечную цель расчета, а построение цепочки модулей и ее выполнение осуществляет сама система, используя при этом сведения о библиотеке модулей, о свойствах понятий из соответствующей области знаний (предметной области) и о возникающих в ней задачах.

Многие ППП с автоматической генерацией программы имеют различные способы представления описания модели предметной области (МПО), а следовательно, и планирования вычислений (таблично-управляемое планирование, условная макрогенерация, планирование на вычислительных моделях и т. д.).

В автоматизированной системе решения прикладных задач со входным языком «Декарт» [1] планирование решения рассматривается как вывод в исчислении предикатов первого порядка. Построение алгоритма решения по описанию МПО и по постановке конкретной задачи в этой модели распадается на несколько этапов:

перевод описания модели и задания на язык формул некоторого формального исчисления 1-го порядка (аксиоматизация),

доказательство существования решения задачи в этом исчислении,

извлечение абстрактной программы из полученного доказательства.

В принципе первый этап не так уж необходим, потому что язык описания моделей и заданий уже достаточно формализован. Возможность поиска доказательства существования решения непосредственно в этом языке (с минимальными дополнениями) будет исследована в дальнейшем. А сейчас, поскольку техника автоматического доказательства теорем хорошо разработана, главным образом, для исчислений 1-го порядка (а метод резолюций требует, кроме того, приведения исходной совокупности формул к специальной стандартной форме), было решено осуществлять упомянутый перевод описаний модели и заданий на язык 1-го порядка, причем сразу к форме, требуемой методом резолюций (см. [2]).

Рассмотрим названные этапы поочередно.

Академия наук Латвийской ССР
Институт экономики
Государственный плановый комитет Латвийской ССР (Госплан)
Научно-исследовательский институт планирования

# ЭКОНОМИЧЕСКАЯ ЭФФЕКТИВНОСТЬ ПРИМЕНЕНИЯ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ В НАРОДНОМ ХОЗЯЙСТВЕ

Республиканский научно-методический семинар
(тезисы докладов)

Рига 1984

# ИЗМЕРЕНИЕ ЭФФЕКТИВНОСТИ ИСПОЛЬЗОВАНИЯ ЭВМ И ПУТИ ЕЕ ПОВЫШЕНИЯ

Балодис Р.П., Борзов Ю.В.,
Брантс Ю.К., Трейманис И.О.
ЛГУ им.П.Стучки

В [I] сказано:
"Постановка отчетности ... есть вещь основная во всех ведомствах и учреждениях самых разнородных".

Ясно, что для определения и повышения эффективности использования парка ЭВМ также необходима объективная отчетность, на основании которой могут приниматься организационно-технические решения, способствующие лучшему использованию машин.

В настоящее время по сути дела единственным официальным показателем эффективности использования ЭВМ является среднесуточный объем произведенного машинного времени. Авторы позволили себе усомниться в достаточной адекватности этого показателя по целому ряду причин: он никак не отражает, с одной стороны, реальную нагрузку на ЭВМ (ясно, что задачи, интенсивно пользующиеся не только центральным процессором, но и периферийным оборудованием, больше "ласмают" машину; высокий коэффициент мультипрограммирования также создает дополнительную нагрузку как на технику, так и на обслуживающий персонал и т.п.), а, с другой стороны, не отражает также реальных "благ", получаемых пользователем ЭВМ (как-то, количество выполненных заданий, среднее время оборота одного задания и т.п.).

В данном докладе сосредоточим внимание на следующих ключевых вопросах:
- показатели и средства учета загрузки ЭВМ;
- показатели и средства учета работы оператора ЭВМ;

- программные средства для повышения эффективности использования ЭВМ;
- РФАП Латвии как средоточие программных средств и методических материалов по вышеуказанным вопросам;
- необходимость резкого качественного повышения обмена новейшей информацией между программистами республики.

## I. Система автоматизированного учета использованных ресурсов ЭВМ.

Автоматизированная система учета машинного времени MAUS [2], вторая, расширенная редакция которой с начала 1984 года эксплуатируется в ЛГУ им.П.Стучки, выдает пользователям отчеты в практически любых разрезах (отдельный пользователь, тема, структурное подразделение организации, группы пользователей, тем, подразделений, ЭВМ и т.п.) за любой отрезок времени по следующим основным показателям:
- время центрального процессора,
- контактное дисплейное время,
- астрономическое время работы ЭВМ,
- коммерческое время (время, начисляемое заданию при его выполнении в мультизадачном режиме),
- стоимость потребленного машинного времени,
- число выполненных заданий,
- число обращений к внешним устройствам,
- объем использованной оперативной памяти,
- коэффициент мультипрограммирования,

и т.д.

## 2. Учет работы оператора ЭВМ.

Число и качество выполненных на ЭВМ заданий в конечном итоге зависит от оператора ЭВМ. Между ним и ЭВМ нет больше посредников. Практика показывает, что на ЭВМ сред-

ней мощности (типа ЕС-1055 или ЕС-1060) при интенсивной разумно спланированной мультипрограммной работе оператора вполне реально достичь двукратного и даже трехкратного увеличения пропуска заданий по сравнению с некоторым интуитивным средним уровнем производительности труда. Однако отсутствие объективных измеримых критериев производительности (ибо количество выполненных заданий из-за весьма большого разброса их параметров не может служить таким критерием) не только не стимулирует труд нерадивых, но (что значительно печальней) отбивает охоту к высокопроизводительному труду даже у самых добросовестных, порождая некую "уравниловку", характеризующуюся не слишком высоким уровнем интенсивности труда. При этом надо отчетливо понимать, что простое повышение зарплаты, не связанное с фактическим повышением интенсивности труда, не только не повышает продуктивность ЭВМ, но еще и морально разлагает некоторых сотрудников.

С начала 1984 года в Вычислительном центре ЛГУ им. П.Стучки проводится эксперимент по автоматизированному учету работы каждого отдельного оператора ЭВМ, а также смен операторов. Информация о производственных показателях собирается специальной подсистемой, состыкованной с системой учета машинного времени MAUS . Учитываются следующие основные показатели:

- число выполненных шагов заданий;
- время работы оператора у ЭВМ;
- произведенное процессорное время;
- произведенное время каналов;
- интегральный показатель произведенной ЭВМ работы: сумма процессорного времени и времени работы каналов с учетом ввода перфокарт, печати, загрузки оперативной памяти и др.технических характеристик;
- коэффициент мультипрограммирования.

Кроме того, все результаты (листинги) проходят контроль системных программистов, и в случае некачественной продукции у оператора ЭВМ снимаются соответствующие браку

количественные показатели.

Описанная нами система учета работы положена также в основу определения результатов социалистического соревнования среди операторов ЭВМ и служит администрации источником объективной информации при решении вопросов об изменении зарплаты, изменении должности, назначении премии и т.п.

## 3. Технологический комплекс программ организации прохождения задач в ЭВМ.

Технологический комплекс ДАУГАВА предназначен для организации вычислительного процесса на ЕС ЭВМ. Он создавался (и продолжает расширяться) в течение почти десяти лет совместными усилиями системных программистов отделов автоматизации программирования и технологии прохождения задач ВЦ ЛГУ им.П.Стучки. Компоненты комплекса представляют собой результаты преодоления "узких мест" в организации вычислительного процесса, например, программа быстрого копирования диска на магнитную ленту и восстановления потребовалась для обеспечения сохранности информации на дисках повышенной емкости (29 и 100 мегабайт), когда над ВЦ ЛГУ им.П.Стучки нависла угроза почти все машинное время употребить на копирование и восстановление дисков. Можно сказать, что комплекс заказан и спланирован самой жизнью. Все компоненты взаимосвязаны, но могут быть использованы также в отдельности.

Вкратце охарактеризуем часть компонент системы ДАУГАВА.

Две из них нами уже приведены ранее – это система MAUS и подсистема OPERATOR.

В комплекс входит визуальная диалоговая система ВДС [3], обеспечивающая работу с локальными дисплеями ЕС-7920 и ЕС-7906. ВДС имеет удобные и наглядные средства редактирования, режим пошагового выполнения ПЛ/I-программ, стыковку с системой MAUS и библиотечной системой COLIBRI,

возможность совмещения диалоговой обработки с параллельным пропуском заданий в пакете.

Библиотечная система COLIBRI обеспечивает приблизительно десятикратную экономию места на магнитном диске, содержащем библиотеки программ (по сравнению со стандартными средствами). Практически на одном 29-мегабайтном диске можно разместить все программы, разрабатываемые коллективом в 200 программистов, причем обеспечивается возможность частого изменения программ без каких-либо дополнительных мер по обслуживанию (т.н. сжатие практически не используется). Тем самым экономятся дисковые пакеты, отпадает необходимость в большом количестве дисководов, значительно сокращается время, употребляемое на копирование дисков с целью обеспечения сохранности информации и, что самое главное, программист в принципе в любой момент может запустить задание на отладку (экономится также время на смену дисков).

Программа быстрого копирования и восстановления дисков LSUDASDR в несколько раз быстрее по сравнению со стандартными утилитами, что экономит время на обслуживание дисков.

Система ARCHIV предназначена для обеспечения хранения на магнитной ленте текстов тех программ, которые давно не изменялись, и быстрого поиска текста по имени программы и программиста с последующим возвратом на диск.

Комплекс включает также средство автоматического учета частоты обращений к разделам библиотек BAS (данные такого учета могут служить, например, поводом для автоматического переноса текста программы в архив). Т.н. бюджетная система предназначается для динамического учета потребленных пользователем ресурсов ЭВМ для информирования оператора ЭВМ об истечении выделенного пользователю на день, сутки или неделю кванта времени и последующего непринятия вычислительной системой задач этого пользователя или назначения им пониженного приоритета. Можно еще отметить средства обеспечения учета времени, потребленного проце -

дурами, запускаемыми с пульта ЭВМ, фиксации контактного дисплейного времени, динамического подключения и отключения дисковых устройств, работающих с диалоговыми системами (в частности, ВДС), специальный набор одношаговых процедур коррекции, трансляции, редактирования и выполнения отлаживаемых программ и ряд других программных средств.

## 4. Республиканский фонд алгоритмов и программ (РФАП Латвии).

Все компоненты технологического комплекса ДАУГАВА либо уже поступили, либо по мере завершения их разработки будут сдаваться в РФАП Латвии вместе с соответствующей документацией. РФАП Латвии передает поступающие в его фонды программные средства заинтересованным организациям на договорной основе. Кроме того, РФАП Латвии содержит также информационные материалы по теоретическим и методическим вопросам разных аспектов программирования и имеет право на их тиражирование.

Авторы призывают активнее использовать услуги РФАП как по распределению программных средств и информационных материалов, так и по их приему для последующего распространения.

РФАП Латвии ведется Вычислительным центром ЛГУ им.П. Стучки. Его адрес -

226250, Рига, ГСП,
бульвар Райниса, 29
ВЦ ЛГУ им.П.Стучки
РФАП Латвии
тел.211014

## 5. О необходимости более тесного общения специалистов.

По мнению авторов ничто так не способствует распространению новейшей научно-технической информации и передовых методов труда, как личные контакты специалистов. В этом плане в области программирования в нашей республике сделано далеко не все возможное. Скажем прямо, что регулярного организованного общения почти нет совсем (если не считать проведенных обществом "Знание" ЛГУ им.П.Стучки циклов публичных лекций "Программирование" и мероприятий Совета молодых специалистов при ЦК ЛКСМ Латвии).

На наш взгляд, в первую очередь необходимо сделать следующее:

- образовать общество программистов или секцию программистов при соответствующем научно-техническом обществе с целью - проведение ежегодных совещаний по проблемам программирования и технологии вычислительных процессов;
- обеспечить программистов республики возможностью публикации материалов по профессиональным вопросам.

Вычислительный центр ЛГУ им.П.Стучки готов взять на себя инициативу в реализации обоих предложений. (Например, публикацию можно обеспечить возможностями РФАП Латвии). Мы предлагаем Вычислительный центр ЛГУ им.П.Стучки как место проведения первого совещания создаваемого общества. Однако мы не в меньшей мере будем приветствовать разные инициативы своих коллег из других организаций.

Литература. I. Ленин В.И. К вопросу о задачах Рабкрина, их понимании и их исполнении. - Полное собрание сочинений. М., издательство политической литературы, 1974, т.44, с.127. 2. Трейманис М.О. MAUS - пакет программ для обработки учетной информации о работе ЭВМ. - Программирование, 1980, № 5, с.89-94. 3. Балодис Р.П. Визуальная диалоговая система программирования. - УСиМ, 1979, № 4, с.51-57.

# МЕТОД НАДЕЖНОГО ТЕСТИРОВАНИЯ ПРОГРАММ ЭВМ

Я.Я.Бичевский, Ю.В.Борзов

Целью отладки программ ЭВМ является проверка совпадения желаемой функции, т.е. функции, которую предположительно должна реализовать отлаживаемая программа, и функции программы, т.е. функции, которую в действительности эта программа реализует. Возможны два пути проверки совпадения названных функций - доказательство правильности программы и проверка правильности программы. Доказательство правильности программы в общем случае представляет собой алгоритмически неразрешимую проблему, а практическое применение этого подхода встречается со значительными трудностями [1]. Проверка правильности программы в большинстве случаев представляет собой выполнение программы на входных данных $X_i$ (тестах) с последующим выяснением правильности результатов $Y_i$. Другими словами, генерируется точка $(X_i, Y_i)$, принадлежащая графику функции программы, и выясняется ее принадлежность графику желаемой функции. Обычно графики рассматриваемых функций содержат бесконечное число различных точек. Автоматические системы SELECT [2], ATTEST [3], CASEGEN [4], СМОТЛ [5], SETAR [6] выбирают множество реализуемых путей, покрывающее весь управляющий граф программы, и затем генерируют тесты $X_i$, на которых эти пути будут пройдены. Хотя таким образом программа проверяется достаточно полно, может оказаться, что сгенерированные тесты и результаты выполнения программы на них недостаточно характеризуют функцию программы. Дело в том, что любой реализуемый путь программы может выполняться не на одном, а на целом множестве тестов. Следовательно, он представляет функцию пути, являющуюся подфункцией функции программы. Само описание функции можно получить символическим выполнением данного пути, что реализовано в системах SELECT [2], ATTEST [3], EFFIGY [7], DISSECT [8]. Объединение функций всех реализуемых путей программы является функцией программы. Однако таких путей, как правило, бесконечно много, что не позволяет при помощи перебора всех путей программы получить описание функции программы. Очевидно, что для автоматического получения описания функции программы следует получить конечное описание всех реализуемых путей программы. В общем случае это алгоритмически неразрешимая проблема; но для программ обработки данных, где, как правило, не производятся сложные преобразования данных, такое описание всех реализуемых путей возможно [9]. А на этом основании возможно и автоматическое построение описания функции программы [10]. Функция программы задается в терминах записей файлов и их составных частей, сопоставляя записям

выходных файлов выражения над соответствующими записями входных файлов. При этом никак не используются понятия переменной и управляющих структур программы. Описание функции программы удовлетворяет свойству: если две программы имеют одинаковые описания, то они функционально эквивалентны.

В настоящее время в ВЦ ЛГУ им.П.Стучки разрабатывается система выделения реализуемых путей ПЛ/I-программ.

В заключение отметим основное отличие нашего подхода от доказательства правильности программы. При доказательстве правильности программы пользователь должен описать желаемую функцию, а затем (автоматически) доказывается, что функция программы совпадает с желаемой функцией. В случае фактического несовпадения остается невыясненным, какую же функцию программа реализует. При нашем подходе по тексту программы (автоматически) получается описание функции программы, которое затем может сравниваться (также автоматически) с описанием желаемой функции, заданным пользователем.

Литература. 1. S.L.Gerhart, L.Yelowitz. Observations of Fallibility in Applications of Modern Programming Methodologies. IEEE Transactions on Software Engineering, vol.SE-2,Sept.1976,pp. 195-207.

2. R.S.Boyer, B.Elspas, K.N.Levitt. SELECT-A Formal System for Testing and Debugging Programs by Symbolic Execution. Proc. 1975 Int.Conf.on Reliable Software, Los Angeles, California, April 1975, pp.234-245.
3. L.A.Clarke. A System to Generate Test Data and Symbolically Execute Programs. IEEE Trans.Software Eng., vol.SE-2, No.3, Sept.1976, pp. 215-222.
4. C.V.Ramamoorthy, S.P.Ho, W.T.Chen. On the Automated Generation of Program Test Data. IEEE Trans.Software Eng., vol. SE-2, No. 4, Dec. 1976, pp. 293-300.
5. J.Bicevskis, J.Borzovs, U.Straujums, A.Zarins, E.F.Miller, Jr. SMOTL-A System to Construct Samples for Data Processing Program Debugging. IEEE Trans.Software Eng., vol.SE-5, No.1, Jan.1979, pp. 60-66.
6. S.Kundu. SETAR-A New Approach to Test Case Generation.INFOTECH State of the Art Report SOFTWARE TESTING, vol.2 : Invited Papers, pp. 161-186.
7. J.C.King. Symbolic Execution and Program Testing. CACM,vol.19, No.7, July 1976, pp. 385-394.
8. W.E.Howden. Symbolic Testing and the DISSECT Symbolic Evaluation System.IEEE Trans.Software Eng.,vol.SE-3,No.4,July 1977,pp. 266-278.

9. Я.М.Барздинь, Я.Я.Бичевский, А.А.Калниньш. Построение полной системы примеров для проверки корректности программ. Ученые записки ЛГУ им.П.Стучки, т.210, Теория алгоритмов и программ, вып.I, Рига, 1974, с.152-167.
10. Я.Я.Бичевский, Ю.В.Борзов. К вопросу о формальном описании функции, реализуемой программой. Тезисы докладов Первой международной конференции молодых ученых "Проблемы проектирования и применения дискретных систем в управлении", Минск, 1977, с.207-208.

# VIII ВСЕСОЮЗНОЕ СОВЕЩАНИЕ ПО ПРОБЛЕМАМ УПРАВЛЕНИЯ

ТАЛЛИН
ОКТЯБРЬ
1980

# ТЕЗИСЫ ДОКЛАДОВ

КНИГА 3

Москва 1980 Таллин

Ю.В.Борзов (Рига)

# АВТОМАТИЧЕСКОЕ ПОСТРОЕНИЕ ОПИСАНИЯ ФУНКЦИИ ПРОГРАММЫ

Любая программа ЭВМ реализует некоторое преобразование входных данных в выходные данные, т.е. вычисляет т.н. функцию программы. Сама программа является одной из возможных форм описания этой функции. Однако такое описание сильно зависит от языка программирования и содержит элементы, неприсущие обрабатываемым данным, например, управляющие структуры, переменные. Соответствие входных данных выходным данным в таком случае остается скрытым.

Известно, что для программ на базовом языке, являющемся упрощенной моделью языков обработки данных (КОБОЛ, ПЛ/I), разрешима проблема автоматического построения полных систем тестов [I]. Оказывается, что для этого класса программ разрешима также проблема автоматического построения описаний функций программы в следующем смысле:

I) Вводится язык для описания преобразований последовательных файлов т.н. язык ФОФ (формул обработки файлов), напоминающий регулярные выражения. Язык ФОФ характерен использованием практически только понятий самих данных - файл, запись, поле записи.

2) Доказывается, что существует алгоритм, который для любой программы на базовом языке строит конечную систему слов на языке ФОФ, задающую в точности то же преобразование данных, что и сама программа.

ЛИТЕРАТУРА. I. Барадинь Я.М., Бичевский Я.Я., Калниньш А.А. Построение полной системы примеров для проверки корректности программ. - В кн.: Теория алгоритмов и программ. Рига, 1974, вып. I, с. 152-188.

Министерство высшего и среднего специального образования
Латвийской ССР
Латвийский ордена Трудового Красного Знамени
государственный университет имени Петра Стучки
Вычислительный центр





# СИНТЕЗ, ТЕСТИРОВАНИЕ, ВЕРИФИКАЦИЯ И ОТЛАДКА ПРОГРАММ

Тезисы докладов
всесоюзной научной конференции

Рига, 22-24 сентября 1981 г.

Латвийский государственный университет им. П.Стучки
Рига 1981

А.А.Калниньш, Ю.В.Борзов (Рига)

## ИНВЕНТАРИЗАЦИЯ ИДЕЙ ТЕСТИРОВАНИЯ ПРОГРАММ

Будем считать, что тестирование – процесс выполнения программы с намерением найти ошибки, а отладка – процесс определения ее местоположения и исправления. Выполнение программы будем понимать в самом широком смысле как любую интерпретацию текста программы. В таком случае тестирование объединяет широкий спектр подходов, на одном конце которого находится традиционное выполнение программы на конкретных входных данных (одиночный тест), а на другом конце – различные методы верификации программы.

Любая синтаксически правильная программа вычисляет какую-то функцию – т.н. функцию программы. Вопрос состоит в том, является ли функция программы желаемой функцией. Желаемая функция описывается на языке спецификаций. Проблема – найти лаконичные, но в то же время точные и понятные языки спецификаций для разных классов задач. Продвижение намечается для задач последовательной обработки файлов с использованием идей реляционной алгебры или расширений понятия конечно-автоматного отображения.

Тестирование традиционно состоит из трех фаз: построения тестов, выполнения программы на тестах, оценки результатов.

Если мы и сможем точно определить желаемую функцию, то для нетривиальных программ никакое конечное множество тестов не может проверить соответствие функции программы желаемой функции. Практически наиболее часто тесты отбираются по критерию $C_1$ : по крайней мере один раз выполняются все реализуемые разветвления в каждом из направлений. При построении тестов решаются две проблемы – найти метод отбора путей, достаточно характерных для программы, и/или найти метод разбиения входных данных на разумное количество классов, характерных для обрабатываемых программой данных. В первом случае чаще всего используются методы статического анализа управляющего графа программы; следует отметить методы разбиения множества путей на классы в

А.А.Калниньш, Д.В.Борзов (Рига)

# ИНВЕНТАРИЗАЦИЯ ИДЕЙ ТЕСТИРОВАНИЯ ПРОГРАММ

Будем считать, что тестирование – процесс выполнения программы с намерением найти ошибки, а отладка – процесс определения ее местоположения и исправления. Выполнение программы будем понимать в самом широком смысле как любую интерпретацию текста программы. В таком случае тестирование объединяет широкий спектр подходов, на одном конце которого находится традиционное выполнение программы на конкретных входных данных (одиночный тест), а на другом конце – различные методы верификации программы.

Любая синтаксически правильная программа вычисляет какую-то функцию – т.н. функцию программы. Вопрос состоит в том, является ли функция программы желаемой функцией. Желаемая функция описывается на языке спецификаций. Проблема – найти лаконичные, но в то же время точные и понятные языки спецификаций для разных классов задач. Продвижение намечается для задач последовательной обработки файлов с использованием идей реляционной алгебры или расширений понятия конечно-автоматного отображения.

Тестирование традиционно состоит из трех фаз: построения тестов, выполнения программы на тестах, оценки результатов.

Если мы и сможем точно определить желаемую функцию, то для нетривиальных программ никакое конечное множество тестов не может проверить соответствие функции программы желаемой функции. Практически наиболее часто тесты отбираются по критерию $C_1$: по крайней мере один раз выполняются все реализуемые разветвления в каждом из направлений. При построении тестов решаются две проблемы – найти метод отбора путей, достаточно характерных для программы, и/или найти метод разбиения входных данных на разумное количество классов, характерных для обрабатываемых программой данных. В первом случае чаще всего используются методы статического анализа управляющего графа программы; следует отметить методы разбиения множества путей на классы в

зависимости от структуры циклов, а также описание всех реализуемых путей в виде конечного графа реализуемости. втором случае плодотворным оказалось использование описа ний входных данных автоматными грамматиками, а также сто хастическая генерация тестов с последующей оценкой полно ты тестированности. Значительным достижением в области генерации тестов следует признать широкое распространени различных модификаций символического выполнения. Введени понятия состояния во многих случаях позволяет получать конечное описание всех реализуемых путей программы, откр вает перспективы разработки языков спецификаций, точно и достаточно наглядно описывающих функцию программы.

Выполнение программы при ее тестировании наряду с па кетным режимом проводится также в диалоговом режиме, кото рый позволяет человеку помочь ЭВМ в решении иначе неразре шимых проблем. Возможно символическое выполнение с получе нием частичных функций программы. Для простых задач обра ботки файлов это позволяет верифицировать программу (про верить соответствие функции программы ее спецификациям). Своеобразным выполнением следует считать статическое тес тирование - т.н. выявление "неправдоподобностей". Интерес ной (и близкой к синтезу) является идея о мутировании про граммы (ее небольшими четко фиксированными изменениями) с целью получения желаемой программы.

Для оценки результатов применяются различные методы накопления информации и ее сравнения с эталонными значени ями. Плодотворной здесь оказалась идея инструментации про граммы или ее выполнения в интерпретирующем режиме. В по следнее время все больше становится програмы, не имеющих четко определенных входных и выходных данных, а лишь пере ходящих из одного состояния в другое. При их тестировании плодотворным может оказаться использование понятия инвари анта.

Все сказанное в основном относится к тестированию отдельного модуля. При тестировании программных систем возникают дополнительные проблемы, связанные с необходи-

мостью создания драйверов, или заглушек, имитирующих работу несуществующих модулей; необходима также генерация данных и сравнение результатов с предполагаемыми, когда информация представляется во внутренней форме системы. Кроме известных методологий нисходящего и восходящего тестирования, надо отметить метод тестовых языков и метод макрокоманд с приоритетами. В тестировании больших программных систем значительную роль играют технологические комплексы программирования.

Известно, что тестирование схем породило богатую теорию булевых функций. Есть основания предполагать, что из тестирования программ возникает не менее интересная теория.

PUBLICĒTS:

Министерство высшего и среднего специального образования Латвийской ССР
Латвийский ордена Трудового Красного Знамени государственный университет имени Петра Стучки
Вычислительный центр

СИНТЕЗ, ТЕСТИРОВАНИЕ, ВЕРИФИКАЦИЯ И ОТЛАДКА ПРОГРАММ

Тезисы докладов
всесоюзной научной конференции

Рига, 22-24 сентября 1981 г.

Латвийский государственный университет им. П.Стучки
Рига 1981

*Р. П. Балодис, Ю. В. Борзов, Ю. К. Брантс, М. О. Трейманис*
(СССР)

# ПУТИ ПОВЫШЕНИЯ ЭФФЕКТИВНОСТИ ИСПОЛЬЗОВАНИЯ ЭВМ

«Постановка отчетности... есть вещь основная во всех ведомствах и учреждениях самых разнородных» [1].

Ясно, что для определения и повышения эффективности использования парка ЭВМ также необходима объективная отчетность, на основании которой могут приниматься организационно-технические решения, способствующие лучшему использованию машин.

В настоящее время, по сути дела, единственным официальным показателем эффективности использования ЭВМ является среднесуточный объем произведенного машинного времени. Авторы позволили себе усомниться в достаточной адекватности этого показателя по целому ряду причин: он никак не отражает, с одной стороны, реальную нагрузку на ЭВМ (ясно, что задачи, интенсивно пользующиеся не только центральным процессором, но и периферийным оборудованием, больше «ломают» машину; высокий коэффициент мультипрограммирования также создает дополнительную нагрузку как на технику, так и на обслуживающий персонал, и т. п.), а с другой стороны, — не отражает также реальных «благ», получаемых пользователем ЭВМ (как-то, количество выполняемых заданий, среднее время оборота одного задания и т. п.).

В одном докладе сосредоточим внимание на следующих ключевых вопросах:

— показатели и средства учета загрузки ЭВМ;

— показатели и средства учета работы оператора ЭВМ;

— программные средства для повышения эффективности использования ЭВМ.

## 1. Система автоматизированного учета использованных ресурсов ЭВМ

Автоматизированная система учета машинного времени MAUS [2], вторая расширенная редакция которой с начала 1984 года эксплуатируется в ЛГУ им. П. Стучки, выдает пользователям отчеты в практически любых разрезах (отдельный пользователь, тема, структурное подразделение организации, группы пользователей, тем, подразделений, ЭВМ и т. п.) за любой отрезок времени по следующим основным показателям:

— время центрального процессора;

— контактное дисплейное время;

— астрономическое время работы ЭВМ;

— коммерческое время (время, начисляемое заданию при его выполнении в мультизадачном режиме);

— стоимость потребленного машинного времени;

— число выполненных заданий;

— число обращений к внешним устройствам;
— объем использованной оперативной памяти;
— коэффициент мультипрограммирования

и т. д.

## 2. Учет работы оператора ЭВМ

Число и качество выполненных на ЭВМ заданий в конечном итоге зависит от оператора ЭВМ. Между ним и ЭВМ нет больше посредников. Практика показывает, что на ЭВМ средней мощности (типа ЕС-1055 или ЕС-1060) при интенсивной разумно спланированной мультипрограммой работе оператора вполне реально достичь двукратного и даже трехкратного увеличения пропуска заданий по сравнению с некоторым интуитивным средним уровнем производительности труда. Однако, отсутствие объективных измеримых критериев производительности (ибо количество выполненных заданий из-за весьма большого разброса их параметров не может служить таким критерием) не только не стимулирует труд нерадивых, но (что значительно печальней) отбивает охоту к высокопроизводительному труду даже у самых добросовестных, порождая некую «уравниловку», характеризующуюся не слишком высоким уровнем интенсивности труда. При этом надо отчетливо понимать, что простое повышение зарплаты, не связанное с фактическим повышением интенсивности труда, не только не повышает продуктивность ЭВМ, но еще и морально разлагает некоторых сотрудников.

С начала 1984 года в Вычислительном центре ЛГУ им. П. Стучки проводится эксперимент по автоматизированному учету работы каждого отдельного оператора ЭВМ, а также смен операторов. Информация о производственных показателях собирается специальной подсистемой, состыкованной с системой учета машинного времени MAUS. Учитываются следующие основные показатели:

— число выполненных шагов заданий;
— время работы оператора у ЭВМ;
— произведенное препроцессорное время;
— произведенное время каналов;
— интегральный показатель произведенной ЭВМ работы: сумма процессорного времени и времени работы каналов с учетом ввода перфокарт, печати, загрузки оперативной памяти и др. технических характеристик;
— коэффициент мультипрограммирования.

Кроме того, все результаты (листинги) проходят контроль системных программистов, и в случае некачественной продукции у оператора ЭВМ снимаются соответствующие браку количественные показатели.

Описанная нами система учета работы положена также в основу определения результатов социалистического соревнования среди операторов ЭВМ и служит администрации источником объективной информации при решении вопросов об изменении зарплаты, изменении должности, назначении премии и т. п.

99–102

### 3. Технологический комплекс программ организации прохождения задач в ЭВМ

Технологический комплекс ДАУГАВА предназначен для организации вычислительного процесса на ЕС ЭВМ. Он создавался (и продолжает расширяться) в течение почти десяти лет совместными усилиями системных программистов отдела автоматизации программирования и технологии прохождения задач ВЦ ЛГУ им. П. Стучки. Компоненты комплекса представляют собой результаты преодоления «узких мест» в организации вычислительного процесса, например, программа быстрого копирования диска на магнитную ленту и восстановления потребовалась для обеспечения сохранности информации на дисках повышенной емкости (29 и 100 мегабайтов), когда над ВЦ ЛГУ им. П. Стучки нависла угроза почти все машинное время употребить на копирование и восстановление дисков. Можно сказать, что комплекс заказан и спланирован самой жизнью. Все компоненты взаимосвязаны, но могут быть использованы также в отдельности.

Вкратце охарактеризуем часть компонент системы ДАУГАВА. Две из них нами уже приведены ранее — это *система* MAUS и подсистема OPERATOR.

В комплекс входит *визуальная диалоговая система ВДС* [2], обеспечивающая работу с локальными дисплеями ЕС-7020 и ЕС-7906. ВДС имеет удобные и наглядные средства редактирования, режим пошагового выполнения ПЛ/1-программ, стыковку с системой MAUS и библиотечной системой COLIBRI, возможность совмещения диалоговой обработки с параллельным пропуском заданий в пакете.

*Библиотечная система* COLIBRI обеспечивает приблизительно десятикратную экономию места на магнитном диске, содержащем библиотеки программ (по сравнению со стандартными средствами). Практически на одном 29-мегабайтном диске можно разместить все программы, разрабатываемые коллективом в 200 программистов, причем обеспечивается возможность частого изменения программ без каких-либо дополнительных мер по обслуживанию (сжатие практически не используется). Тем самым экономятся дисковые пакеты, отпадает необходимость в большом количестве дисководов, значительно сокращается время, употребляемое на копирование дисков с целью обеспечения сохранности информации и, что самое главное, программист, в принципе, в любой момент может запустить задание на отладку (экономится также время на смену дисков).

*Программа быстрого копирования и восстановления дисков* LSUDASDR в несколько раз быстрее по сравнению со стандартными утилитами, что экономит время на обслуживание дисков.

*Система* ARCHIV предназначена для обеспечения хранения на магнитной ленте текстов тех программ, которые давно не изменялись, и быстрого поиска текста по имени программы и программиста с последующим возвратом на диск.

Комплекс включает также *средство автоматического учета частоты обращений к разделам библиотек* BAS (данные такого учета могут служить, например, поводом для автоматического переноса текста программы в архив). Так называемая б ю д ж е т н а я с и с т е м а предназначается для динамического учета потребленных пользователем ресурсов ЭВМ для информирования оператора ЭВМ об истечении выделенного пользователю на день, сутки или неделю кванта времени и последующего непринятия вычислительной системой задач этого пользователя или назначения им пониженного приоритета. Можно еще отметить средства обеспечения учета времени, потребленного процедурами, запускаемыми с пульта ЭВМ, фиксации контактного дисплейного времени, динамического подключения и отключения дисковых устройств, работающих с диалоговыми системами (в частности, ВДС), специальный набор одношаговых процедур коррекции, трансляции, редактирования и выполнения отлаживаемых программ и ряд других программных средств.

СПИСОК ЛИТЕРАТУРЫ

1. Л е н и н В. И. К вопросу о задачах Рабкрина, их понимании и их исполнении. — Полное собрание сочинений. М.: Полит. лит-ра, 1974, т. 44, с. 127.

2. Б а л о д и с Р. П. Визуальная диалоговая система программирования. — УСиМ, 1979, № 4, с. 51—57.

3. Т р е й м а н и с М. О. MAUS — пакет программ для обработки учетной информации о работе ЭВМ. — Программирование, 1980, № 5, с. 89—94.

PUBLICETS:

МЕЖДУНАРОДНАЯ
НАУЧНО-ТЕХНИЧЕСКАЯ
КОНФЕРЕНЦИЯ
«ПРОГРАММНОЕ
ОБЕСПЕЧЕНИЕ ЭВМ»

Секция 4

# ЭКОНОМИЧЕСКАЯ И СОЦИАЛЬНАЯ ЭФФЕКТИВНОСТЬ ПРИМЕНЕНИЯ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Министерство высшего и среднего специального образования
Латвийской ССР
Латвийский ордена Трудового Красного Знамени
государственный университет имени Петра Стучки
Вычислительный центр

# МЕТОДИКА СОЗДАНИЯ МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ ЭВМ

Межвузовский сборник научных трудов

Под общей редакцией А.Калниньша



Латвийский государственный университет им. П.Стучки
Рига 1980

# ОДИН ПОДХОД К ОТЛАДКЕ БОЛЬШИХ СИСТЕМ

Я.Я.Бичевский, Ю.В.Борзов

ВЦ ЛГУ им.П.Стучки

## I.Основные идеи

Мы рассмотрим один подход к отладке больших систем, который был применен при разработке системы СМОТЛ [I]. Суть этого подхода состоит в следующем.

Строится специальная система отладки (СО), которая является составной частью разрабатываемой системы. Основным средством СО служат макрокоманды, вставляемые в отлаживаемые программы. Притом имеется возможность управлять выполнением макрокоманд: каждая макрокоманда имеет свой приоритет, задаваемый программистом при ее написании. Кроме этого, имеется так называемый текущий приоритет системы, который в каждом сеансе задается извне. Перед выполнением каждой макрокоманды ее приоритет сравнивается с текущим приоритетом системы. Если приоритет макрокоманды меньше или равен текущему приоритету системы, то она выполняется, в противоположном случае - игнорируется.

Таким образом, имеется возможность без каких-либо изменений системы в различных сеансах осуществлять различные режимы отладки. В частности, можно проигнорировать все макрокоманды отладки - эксплуатировать систему в реальном режиме.

## 2. Ситуация и проблемы

Система СМОТЛ разрабатывалась в 1974-1976 гг. сотрудниками ВЦ Латвийского госуниверситета Василевским М.П., Зариньшем А.К., Страуюмсом У.М. и авторами данной статьи. Система предназначена для автоматического построения полных систем примеров на ЭВМ "Минск-32". Отличительными осо-

бенностями СМОТЛ являются:

- алгоритмы весьма сложны и зачастую носят эвристический характер;

- системе необходимо выполнить работу большого объема (неоднократный просмотр всех путей больших графов и т.п.), следовательно, программы должны быть максимально быстродействующими,

- вся обрабатываемая информация представлена в виде списков.

Функционально система СМОТЛ состоит из управляющего блока (УБ) и десяти рабочих блоков (РБ). (Смотрите рисунок I, на котором передача управления изображена непрерывными, а передача информации - прерывистыми стрелками).

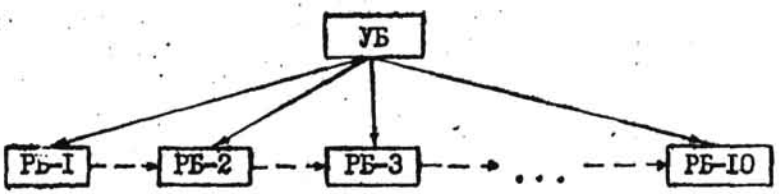



Рис.I.

Система управляется заказом, который вводится с перфокарт. УБ является резидентом системы; он загружает в оперативную память рабочие блоки и передает им управление. Входной информацией для каждого РБ является результат работы предыдущего РБ. Эта информация в виде сложного списка хранится в оперативной памяти. Итак, отметим проблемы, возникшие перед нами:

I. Применимость эвристических алгоритмов проверяется практикой. Только запуская всю систему, мы можем определить пригодность выбранных алгоритмов. Это заставляет нас начать комплексную отладку системы по возможности раньше.

2. Ранняя комплексная отладка требует предусмотреть возможность запуска всей системы при отсутствии некоторых её частей и имитацию работы этих частей. (Идеи об имитации работы отсутствующих блоков почерпнуты из [2]).

3. Ввиду того, что алгоритмы носят эвристический характер, их недостатки могут выявляться в опытной и даже в промышленной эксплуатации. Поэтому необходимо ориентироваться на ситуацию, когда система будет несколько раз изменяться и отлаживаться повторно.

4. При разработке системы должна обеспечиваться возможность получения отладочной информации различной степени детализации. Притом для получения этой информации во время сопровождения системы изменять программы уже недопустимо. Отладочная информация должна быть получена очень простым путем, например, изменением нескольких карт в заказе на работу системы.

## 3. Неприменимость стандартных средств отладки

Стандартное средство отладки программ - ОП-2, доступное в 1974 году, при всех его достоинствах не удовлетворяет указанным выше требованиям, вытекающим из специфики нашей работы, а именно:

- Степень замедления работы системы, выполняемой под управлением ОП-2, в нашем случае неприемлема. (Отметим только то, что СМОТЛ зачастую использует до 10 минут процессорного времени, а под управлением ОП-2 это составляло бы несколько часов).

- Хотя ОП-2 имеет возможность записи информации в оперативную память и ее распечатки, однако не предполагается работа со списочными структурами данных. (Для задания достаточно простого графа нам потребовалось бы несколько сотен перфокарт, не говоря уже о многочасовом кропотливом труде по кодированию информации в виде графа).

- Ввиду того, что ОП-2 занимает часть оперативной памяти, в принципе невозможно воспроизвести ту же самую ситуацию, которая возникает при работе системы без использования ОП-2.

- Возникают трудности, связанные с выдачей отладочной информации различной степени детализации различными блока-

ми системы. При использовании ОП-2 заказ на отладку ввиду его сложности может составлять лишь разработчик конкретного блока. Это существенно усложняет комплексную отладку системы и получение отладочной информации на стадии опытной эксплуатации системы.

Отметим, что указанные выше проблемы, за исключением, быть может, первой, не решают также отладочные программы ОП-3 для ЭВМ "Минск-32" и ТЕСТРАН для ЕС ЭВМ.

Поэтому для разработки системы СМОТЛ было принято решение разработать свою специальную СО.

## 4. Решение проблем

Система отладки реализована следующим образом. В рабочих блоках в виде макрокоманд вставлены обращения к программе, реализующей отладочные операции. Эту программу будем называть отладочным блоком (ОБ). В нашей системе ОБ физически включен в управляющий блок и при работе системы СМОТЛ находится в оперативной памяти. Объем ОБ — 2300 ячеек, включая модули ввода-вывода. В макрокомандах задавались следующие отладочные действия:

1. Ввод с перфокарт закодированного произвольного списка и его размещение в оперативной памяти. Адрес размещения начала списка помещается в переменную, указанную в макрокоманде.

2. Вывод на печать списка, адрес которого указан в переменной, которая, в свою очередь, задается в макрокоманде.

Каждая макрокоманда содержит специальный операнд, называемый приоритетом макрокоманды. Значение этого операнда задает программист при написании макрокоманды, и оно в дальнейшем не меняется. В начале выполнения каждой макрокоманды приоритет этой макрокоманды сравнивается с текущим приоритетом системы. Если приоритет макрокоманды ниже или равен приоритету системы, то макрокоманда выполняется. В противном случае она не выполняется. Приоритет системы задается управляющей картой в заказе на выполнение СМОТЛ. В этой

карте каждому рабочему блоку присваивается определенный приоритет блока (в нашем случае-от 0 до 7). Нулевой приоритет блока означает, что соответствующий рабочий блок в данном сеансе вообще не выполняется (даже не загружается в оперативную память и при работе системы он вообще может отсутствовать). Остальные приоритеты блока означают, что соответствующий рабочий блок должен быть выполнен и при его выполнении текущий приоритет системы устанавливается равный приоритету этого блока. Например, если в данном сеансе третий рабочий блок имеет приоритет 4, то УБ после выполнения первых двух РБ загружает в оперативную память третий РБ, устанавливает текущий приоритет системы, равный 4, и передает управление третьему РБ. В течение работы третьего блока будут выполнены все те макрокоманды отладки, которые имеют приоритет от I до 4, а макрокоманды с приоритетами от 5 до 7, хотя и они в программе могут присутствовать, выполнены не будут.

Ясно, что при разумном использовании приоритетов макрокоманд отладки в зависимости от заказа на выполнение системы СМОТЛ можно получить отладочную информацию различной степени детализации.

Макрокоманды отладки нет нужды менять, и они могут присутствовать в готовой системе. На быстродействие системы СМОТЛ это влияет незначительно, потому, что сравнение приоритета макрокоманды и текущего приоритета системы осуществлено несколькими машинными командами. В частности, полезно оставлять в начале каждого РБ макрокоманды отладки, распечатывающие входную информацию. (Мы им присваивали приоритет 2.) Это намного облегчит в будущем локализацию ошибок.

Отлаженные РБ обычно выполнялись с приоритетом I, а макрокоманды с приоритетом I вообще не использовались. Это позволяло нам выполнить отлаженные блоки без каких-либо отладочных действий.

Теперь рассмотрим имитацию работы несуществующих или неотлаженных РБ. Имитация реализована не замещением РБ мо-

делирующей программой (как в [2]), а включением макрокоманды отладки в начало следующего выполняемого РБ. (При этом имитируемый РБ вообще не выполняется, т.е. ему присваивается нулевой приоритет блока).

Если при выполнении РБ приоритет этой макрокоманды ниже или равен текущему приоритету системы, то она создает в оперативной памяти список, который соответствует результату работы предыдущего (но фактически невыполненного) РБ. (В нашем случае этот список в закодированном виде вводился с перфокарт.) Таким образом получается, что каждый РБ может создавать входные данные для себя и отлаживаться автономно.

Если при выполнении РБ приоритет макрокоманды отладки выше текущего приоритета системы, то она не выполняется. Следовательно, РБ будет обрабатывать информацию, в действительности созданную предыдущим РБ.

Ясно, что этот метод имитации позволяет легко переходить от автономной отладки РБ к комплексной и обратно без всякого изменения разрабатываемой системы: меняется только одна карта, задающая приоритеты выполнения РБ в заказе на работу системы.

Нетрудно заметить, что использование одного и того же текущего приоритета системы для двух различных операций ввода и вывода информации может оказаться в некоторых случаях неудобным. Фактически для каждого типа макрокоманд необходимо было бы задавать текущий приоритет системы независимо. В таком случае значением текущего приоритета системы было бы не одно число, а пара чисел - первое число будет приоритетом ввода, второе - приоритетом вывода. В системе СМОТЛ это не реализовано, однако имеется возможность одной управляющей картой заказа отключить полностью выполнение или всех макрокоманд ввода или всех макрокоманд вывода.

## 5. Результаты

Разработка описанного выше отладочного блока потребовала примерно 4 человеко-месяца. Но его использование, как оказалось позже, сэкономило, по нашим расчетам, по крайней мере, 3 человеко-года. Дело в том, что разработка и отладка всех РБ проводилась параллельно. Комплексную отладку удалось начать уже через 3 месяца с начала разработки рабочих блоков. Кроме этого, один из РБ не удавалось отладить в течение полугода после того, как все остальные РБ уже функционировали. Несмотря на это, система отладки давала возможность имитировать работу этого РБ, а остальные блоки выполнять в реальном режиме.

Оставленные в РБ макрокоманды отладки позволяют эффективно сопровождать систему.

## 6. Применимость подхода

Большинство АСУ, транслирующих систем и т.п. имеют свои управляющие блоки и языки управления системой, наличие которых существенно для реализации описанного метода отладки. Следовательно, изменив макрокоманды отладки с учетом специфики обрабатываемой информации, рассмотренный подход вполне применим в более широких масштабах. Свидетельство тому - успешное его использование при разработке системы диалоговой отладки ПЛ/I-программ и системы АИСК в ВЦ ЛГУ им.П.Стучки в 1977-1979 гг.

## ЛИТЕРАТУРА

1. Bičevskis J., Borzovs J., Straujums U., Zariņš A., Miller E.F. Jr.SMOTL - A System to Construct Samples for Data Processing Program Debugging. - IEEE Transactions on Software Engineering, 1979, vol.SE-5, No.1, p.60-66.
2. Miller E.F., Lindamood G.E.Structured Programming: Top-down Approach. - Datamation, 1973, vol.19, No.12, p.55-57.

# ПРОСТОЙ МЕТОД ОБЕСПЕЧЕНИЯ ДИАЛОГОВОЙ ОТЛАДКИ ПЛ/I - ПРОГРАММ

Ю.В.Борзов

## I. Основная идея.

В текст отлаживаемой программы автоматически вставляются обращения к отлаживающей программе, которая связывает отлаживаемую программу с дисплеем и таким образом обеспечивает диалоговый режим выполнения.

## 2. Возможности для пользователя.

Легко реализовать следующие возможности для пользователя

- передача управления на любой оператор программы с последующим возвратом управления пользователю, если достигнут указанный пользователем оператор или выполнено указанное пользователем число операторов, или же возникла авария;
- вывод текущего значения переменной;
- присвоение заданного пользователем значения переменной
- вывод трассы (последовательности номеров выполненных операторов) программы;
- проверка выполнения заданных пользователем отношений между значениями переменных;
- выдача профиля программы пользователя (списка чисел выполнения каждого оператора) в конце сеанса отладки.

## 3. Инструментация программы.

Перед каждым выполняемым оператором программы пользователя автоматически вставляется обращение к отлаживающей программе и метке. Если, например, в исходной программе пользователя есть оператор A:B=C; с номером 5, проставленным ПЛ/I компилятором, тогда после инструментации вместо этого оператора будет проставлено

```
A: CALL #B(5); #(5): B=C;
```

где #B - отлаживающая программа. Параметр процедуры #B (в данном случае - 5) используется для того, чтобы отлаживаю-

мая программа "знала" номер оператора, который будет выполнен следующим. Метка (в данном случае -#(5)) используется для того, чтобы из отлаживающей программы имелась возможность передать управление на любой оператор программы пользователя.

4. Отлаживающая программа.

Отлаживающая программа является внутренней процедурой инструментированной программы пользователя. Поэтому передачу управления из отлаживающей процедуры в отлаживаемую программу можно реализовать оператором GO TO. Например, если пользователь желает передать управление на оператор с номером 8, то в отлаживающей процедуре надо выполнить фрагмент:

```
...
I = 8;
GO TO #(I);
...
```

Выполнение следующего оператора программы пользователя реализуется оператором RETURN в отлаживающей процедуре.

Для того, чтобы узнать значение переменной, используется оператор PUT STRING ( ) DATA ( ) ;. Например, пользователь хочет узнать значение переменной M3.

```
...
DCL M3 FIXED (3);
DCL A CHAR(32Ø) VAR;
PUT STRING (A) DATA (M3);
...
```

После выполнения такого фрагмента в переменной A окажется, например, 'M3=222', после чего это значение будет показано на экране дисплея.

Для присвоения значения переменной используется оператор GET STRING ( ) DATA ( ) ;. Например, пользователь с дисплея хочет задать значение переменной M3 равное 222. Для этого значение 'M3=222' заносится в переменную A, после чего выполняется фрагмент программы

```
...
DCL M3 FIXED (3);
DCL A CHAR (32Ø) VAR;
GET STRING (A) DATA (M3);
...
```

После выполнения такого фрагмента значение переменной M3 будет равным 220.

Проверка соотношений между значениями переменных проходит при помощи выполнения в отлаживающей процедуре соответствующих IF -операторов, а профиль и трасса программы накапливается в массивах, определенных в отлаживающей процедуре.

## 5. Функциональная схема отладки программы пользователя.

Шаг I. Инструментация программы пользователя.

Шаг 2. Генерация отлаживающей процедуры в зависимости от заказа пользователя и ее присоединение к программе пользователя.

Шаг 3. Компиляция и редактирование связей объединенной программы в пакетном режиме.

Шаг 4. Выполнение объединенной программы в диалоговом режиме.

## 6. Машинный эксперимент.

В 1978/79 гг. в Вычислительном центре Латвийского государственного университета им.П.Стучки под руководством автора была проведена машинная проверка описанного метода. Дипломантка Р.В.Звирбуле, студентка 4-го курса И.Т.Розенштейне и В.В. Сестуле разработали экспериментальную диалоговую систему отладки ПЛ/I-программ. Программирование системы было также проведено на ПЛ/I. Получены обнадеживающие результаты. Скорость выполнения программы пользователя в диалоговом режиме достигала порядка 400 операторов ПЛ/I в секунду на ЭВМ ЕС-1022. Занимаемая оперативная память увеличивалась по сравнению с обычным выполнением программы пользователя в допустимых пределах - на 20К байт плюс 3К байт на каждые 100 операторов программы пользователя. Более детальные показатели получены не были.

## 7. Текущее состояние разработки.

В данный момент дипломантки И.Т.Розенштейне и В.В.Сестуле продолжают развитие системы. Реализуются возможности проверки утверждений, получения трассы и профиля, не включенные в экспериментальную разработку. Предполагается включение данной системы в систему ДУВЗ (Диалоговый Удаленный Ввод Заданий). Эти системы взаимно дополняют друг друга, так как ДУВЗ не имеет

возможности диалогового выполнения ПЛ/I-программ, но имеет средства редактирования программ. В качестве эксперимента запланировано создание обучающей подсистемы, обучающей пользованию системой отладки в диалоговом режиме. Предполагаемый срок завершения разработок – май 1980 г.

PUBLICETS.

ГРУЗИНСКИЙ РЕСПУБЛИКАНСКИЙ СЕМИНАР
«ЧЕЛОВЕКО-МАШИННЫЕ СИСТЕМЫ»

# ИНТЕРАКТИВНЫЕ СИСТЕМЫ

## ТЕЗИСЫ ДОКЛАДОВ ВТОРОЙ ШКОЛЫ-СЕМИНАРА

(Боржоми, февраль, 1980 г)

I книга

«МЕЦНИЕРЕБА»

19 ТБИЛИСИ 80

Ю.В.Борзов, /Рига/, Г.Э.Дишлерс /Рига/, Л.Б.Дулькин /Минск/, Г.В.Ермаков /Минск/, Т.В.Ермакова /Минск/, В.Л. Катков /Минск/, И.Э.Медведис /Рига/, Г.Б.Уртанс /Рига/

# ЭКСПЕРИМЕНТЫ С СИСТЕМОЙ СИМВОЛЬНОГО ТЕСТИРОВАНИЯ ПЛ/I-ПРОГРАММ

ТЕСТГЕН является экспериментальной системой символьного тестирования /1/ СИПЛ/I-программ. СИПЛ/I /2/ – подмножество ПЛ/I, дополненное фильтрацией /3/. ТЕСТГЕН обеспечивает автоматический отбор множества реализуемых путей программы согласно критерию $C_1$, построение тестов и функций для этих путей и т.д. Применяются эвристические алгоритмы /4/, используется опыт теоретических исследований /5,6/ и практической реализации /7/.

Приводятся данные о функционировании системы.

ЛИТЕРАТУРА. 1. Борзов Ю.В. Тестирование программ с использованием символического выполнения// Программирование.– 1980.– № 1.– С.51–59. 2. Катков В.Л., Пилецкий И.И. Подмножество языка ПЛ/I для системного программирования// УСиМ – 1983.– № 3 – С.59–63. 3. Марголин М.С. О фильтрации параметров программных объектов// Программирование.– 1978.– № 4 – С.35–37. 4. Борзов Ю.В., Дишлерс Г.Э., Медведис И.Э., Уртанс Г.Б. Основные алгоритмы символического тестирования программ// РФАП Латвии – 1983.– № ИМООІО – С.28. 5. Барздинь Я.М., Бичевский Я.Я., Калиньш А.А. Построение полной системы примеров для проверки корректности программ// Уч.зап.Латв.гос.ун-та – 1974. – Т.210 – С.152–187. 6. Борзов Ю.В., Уртанс Г.Б. Формулы обработки файлов// Латв.матем.ежегодник – 1983.– Вып. 27 – С.202–209. 7. Bicevskis J., Borzovs J., Straujums U., Zarins A., Miller E.F.,Jr. SMOTL – A System to Construct Samples for Data Processing Program Debugging// IEEE Trans.Software Eng. – 1979 – SE-5 – N 1 – P.60–66.

Министерство высшего и среднего специального
образования Латвийской ССР
Латвийский ордена Трудового Красного Знамени
государственный университет им. П.Стучки
Вычислительный центр

# ПРОБЛЕМЫ СОВЕРШЕНСТВОВАНИЯ СИНТЕЗА, ТЕСТИРОВАНИЯ, ВЕРИФИКАЦИИ И ОТЛАДКИ ПРОГРАММ

Тезисы докладов
Всесоюзной научной конференции
Рига, 12-14 ноября 1986 г.
Том I

Второе стереотипное издание





Латвийский государственный университет им. П.Стучки
Рига 1987

# АДЕКВАТНО ЛИ СИМВОЛЬНОЕ ВЫПОЛНЕНИЕ ?

Ю.В.Борзов

Символьное выполнение пути программы означает алгоритмическое сопоставление произвольному пути программы корректного символьного состояния.

Символьный язык назовем алгоритмически полным по отношению к языку программирования, если существует алгоритм, сопоставляющий произвольному пути любой программы корректного символьного состояния.

Назовем язык программирования однозначным, если существует алгоритм, который, используя только текст программы и значения всех ее переменных до выполнения произвольного оператора, детерминированно вычисляет значения всех переменных после выполнения этого оператора и определяет следующий выполняемый оператор.

УТВЕРЖДЕНИЕ. Язык программирования имеет алгоритмически полный символьный язык тогда и только тогда, если он является однозначным.

Все широко применяемые языки программирования не являются однозначными.

СЛЕДСТВИЕ. Все широко применяемые языки программирования не имеют алгоритмически полного символьного языка.

## Литература

1. Костырко В.С. О соотношении методов проверки правильности программ. - Кибернетика, 1988, № 4.

2. Борзов Ю.В. Тестирование программ с использованием символического выполнения. - Программирование, 1980, № 1.

АКАДЕМИЯ НАУК СССР
ГОСУДАРСТВЕННЫЙ КОМИТЕТ СССР ПО НАРОДНОМУ ОБРАЗОВАНИЮ
НАУЧНЫЙ СОВЕТ АН СССР ПО ПРОБЛЕМАМ УПРАВЛЕНИЯ ДВИЖЕНИЕМ И НАВИГАЦИИ
СЕКЦИЯ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СИСТЕМ УПРАВЛЕНИЯ
ХАРЬКОВСКИЙ ОРДЕНА ЛЕНИНА АВИАЦИОННЫЙ ИНСТИТУТ им. Н. Е. ЖУКОВСКОГО

# ВТОРОЕ ВСЕСОЮЗНОЕ СОВЕЩАНИЕ ПО АВТОМАТИЗИРОВАННОМУ ПРОЕКТИРОВАНИЮ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СИСТЕМ УПРАВЛЕНИЯ ДВИЖУЩИМИСЯ ОБЪЕКТАМИ

25—29 сентября

*ТЕЗИСЫ ДОКЛАДОВ*




Харьков
1989

# ОБЕСПЕЧЕНИЕ ОБРАБОТКИ ЛАТЫШСКОГО ТЕКСТА НА ЭВМ

Ю.В.Борзов, к.ф.-м.н. А.Р.Спектор,
к.ф.-м.н. Н.Н.Устинов

Постепенное расширение делопроизводства на латышском языке и необходимость обучения информатике на родном языке неизбежно наталкивается на неприспособленность к нему используемой в Латвии вычислительной техники. Отечественная вычислительная техника, как правило, обеспечивает обработку текстов только на русском и английском языках, причем ее приспособление к латышскому языку требует конструктивных доработок в аппаратуре. Новейшая импортная техника и перспективные советские ЭВМ в той или иной мере позволяют изменять алфавит программным путем, однако, для удобного переноса информации с машины на машину необходимо соблюдать определенные стандарты кодирования. Таких стандартов или даже рекомендаций до сих пор не имелось.

В первую очередь необходимо решить нижеперечисленные вопросы.

- Разработка клавиатуры для ввода латышского текста. Для этого, видимо, нужно спроектировать и обеспечить производство одно- и двухалфавитных клавиатур (латышско-латгальско-английской и русской) с эргономическим размещением знаков латышского алфавита. Заметим, что эргономическая латышская клавиатура необходима также для пишущих машинок и разного рода оргавтоматов.
- Разработка кодовых таблиц для внутреннего кодирования информации в ЭВМ, обеспечивая максимально возможную совместимость с существующими стандартами с целью использования накопленного в мире программного обеспечения.
- Разработка шрифта и техническое обеспечение выдачи информации на дисплей и печатающие устройства в латышском алфавите.

Данными вопросами занимается межведомственная комиссия при Госплане Латвийской ССР. Головной организацией по проблеме является ВЦ при ЛГУ им.П.Стучки. Рекомендации, разработанные комиссией, приводятся в докладе.

Академия наук Латвийской ССР
Институт экономики

Государственный плановый комитет Латвийской ССР
Научно-исследовательский институт планирования

Государственный комитет СССР по вычислительной технике и информатике
Центральный экономико-математический институт АН СССР

# ЭКОНОМИКА КОМПЬЮТЕРИЗАЦИИ НАРОДНОГО ХОЗЯЙСТВА

Тезисы докладов
Межреспубликанского научно-практического семинара

В 2-х томах

Том I





Рига 1989

J.Borzovs
R.Čevere
M.Lučkina

# DATORA IZMANTOŠANAS PRAKSE LZA TERMINOLOĢIJAS KOMISIJAS INFORMĀTIKAS APAKŠKOMISIJAS DARBĀ

Varētu domāt, ka pēc tam, kad vairāk vai mazāk noskaidroti terminoloģijas izstrādes vispārējie [1] un nozares konkrētie [2] principi, atliek vairs tikai rutīnas darbs vajadzīgo terminu radīšanā. Pat neņemot vērā to, ka minētie principi nav gluži savstarpēji nepretrunīgi un nav arī vienīgie iespējamie, tomēr pastāv vēl arī vairāki citi grūti risināmi praktiski jautājumi, ko vēl tikai padziļina pieaugošais jau apstiprināto terminu apjoms. Citu starpā tie ir:

- apstiprināto terminu un to skaidrojumu, kā arī apstiprināšanas vēstures uzturēšana;
- jaunveidojamo terminu saskaņošana ar jau esošajiem terminiem;
- citu nozaru terminu respektēšana nozaru saskarapgabalos;
- iespējami plaša terminu un to skaidrojumu projektu apspriešana pirms to apstiprināšanas;
- latviešu terminoloģijas kopbāzes izveidošana un uzturēšana;
- vārdnīcu sagatavošana izdošanai.

Informātikas apakškomisija jau vairākus gadus izmanto personālo datoru un īpaši izstrādātas datorprogrammas [3] minēto jautājumu risināšanai. .

- Nesen izdotā vārdnīca [4] tika gatavota iespiešanai, izmantojot tikai personālā datora iespējas. Īpašas programmas palīdzēja veikt rūpīgas un visaptverošas šķērspārbaudes, kas ļāva novērst lielu daudzumu neuzmanības kļūdu, kā arī viegli sagatavot latviešu un krievu terminu rādītājus (vārdnīca sakārtota angļu terminu secībā).
- Visa pašreizējā informācijas tehnoloģijas apstiprināto terminu kopa glabājas vienā noteiktā datorā, un to uzturēt (papildināt, labot u.tml.) drīkst tikai viens pilnvarots speciālists, toties šīs terminu kopas kopijas tiek brīvi izplatītas datorfildos (tās iespējams saņemt arī uz diskešm SWH IS Komerccentra salonā un pie pārstāvjiem rajonu pilsētās). Domstarpību gadījumos par etalonu tiek atzīta jau minētajā datorā glabātā terminu kopa, kurā izmaiņas drīkst izdarīt, pat acīm redzamu kļūdu gadījumā, tikai ar apakškomisijas dokumentētu lēmumu. Pēkšņi labojumi terminu apspriešanas gaitā ne tikai nav pieļaujami, bet ir tehniski neiespējami. Tas ļoti disciplinē darbu un nodrošina esošās terminu bāzes integritāti.
- Apspriežot kārtējo jauno terminu, tiek izmantots dators, kurā atrodas svaigākā etalona kopija. Izmantojot jau minētās programmas, tiek veikta apspriežamā termina pārbaude, kas pat visai vāja datora gadījumā neaizņem vairāk par dažiem desmitiem sekunžu (terminu bāzē pašlaik ir pāri par 3000 terminu), turklāt šo pārbaudi var veikt, nepārtraucot apspriešanu. Tā tiek panākts, ka dažādiem angļu terminiem netiek piekārtots viens un tas pats latviešu termins, ka viens un tas pats latviešu vārds netiek izmantots atšķirīgu jēdzienu apzīmēšanai u.tml.

- Lai terminu projektu iepriekšējā apspriešanā iesaistītu iespējami plašu interesentu loku, jo ne visi atrod laiku piedalīties apakškomisijas sēdēs, kaut tās visas ir atklātas un izteikties un balsot atļauts ikvienam klātesošam, elektroniskā formā sagatavotie priekšlikumi tiek darīti pieejami caur t.s. Interneta globālo tīmekli (WWW - World Wide Web). Ikviens interesents var savus priekšlikumus darīt zināmus apakškomisijai - personiski, telefoniski vai izmantojot epastu.

Būtu vēlams arī citām apakškomisijām izmantot šeit minētās procedūras, kas dotu iespēju uzturēt latviešu terminu kopbāzi saskaņotā formātā, kā arī izmantot "kaimiņu" terminus robežapgabalos, kur saskaras vairāku nozaru jēdzieni. Informācijas tehnoloģijā tāda saskare visvairāk ir ar fiziku, matemātiku un poligrāfiju.

Valrums izmantoto procedūru ir ārkārtīgi vienkāršas un pat pilnīgam datora nezinātājam viegli apgūstamas dažu desmitu minūšu laikā.

Rīgas Informācijas tehnoloģijas institūts būtu ar mieru līdztekus informācijas tehnoloģijas terminiem uzturēt arī visu latviešu terminu kopbāzi, kā arī sniegt nepieciešamās konsultācijas.

**Atsauces**


1. V.Skujiņa. Latviešu terminoloģijas izstrādes principi._ Rīga, Zinātne._ 1993.
2. J.Borzovs. Lai iegūst latviešu valodu!_ Datortehnika, Nr.3._ 1993._lpp.7-8.
3. R.Čevere, J.Borzovs. Lietotāja pamatprasības tipveida darba seansiem ar datorizētu terminoloģisko vārdnīcu._ Baltistica VII, VII_Starptautiskais baltistu kongress._ Rīga._ 1995._ lpp.25-27.
4. Angļu-krievu-latviešu skaidrojošā vārdnīca: Datu pārraides un apstrādes sistēmas. Sast. V.Baumgarts, A.Baums, A.Gobzemis, G.Fricnovičs, Ilziņa, terminol. konsult. V.Skujiņa._ Rīga, a/s SWH informatīvās sistēmas._ 1995.


PUBLICĒTS:

Latvijas Zinātņu akadēmija
Latviešu valodas institūts

# Nacionālā terminoloģija: vēsture, tagadne un perspektīvas

**Referātu tēzes**

Latvijas Zinātņu akadēmijas un
LZA Terminoloģijas komisijas
50-gadei veltītā konference
un
Lietuvas, Igaunijas un Latvijas terminologu
apaļais galds

## Nacionālās terminoloģijas normalizācija un starptautiskā sadarbība

14.-15.10.1996.
Rīga

Обсуждаемые элементарные средства реализованы на языке МОДУЛА-2 в системе РАФОС. Программы весьма лаконичны, например объектный модуль символьного процессора занимает семь блоков. Все средства ориентированы на программиста — разработчика ППП в конкретной предметной области.

Отметим, что даже в простейших случаях, аналогичных рассмотренному выше (модель TLF1), количество операторов в описании задачи оказывается в 8—10 раз меньше количества вызовов процедур в плане, построенном по этому описанию. При этом любой из вызовов процедур конструирования *I*-объектов связан с заданием множества атрибутов — параметров модели. Поэтому, на наш взгляд, одноразовый труд по определению соответствующего ЯС многократно окупается удобствами пользователя ППП, который получает возможность мыслить в категориях ЯС, а не в конструкциях вызовов параметризованных ППП.


1. *Фути К., Судзуки Н.* Языки программирования и схемотехника СБИС. — М.: Мир, 1988. — 224 с.
2. *Stanley Z.* Why properties are objects some refinements of "is-a" // Proc. Fall Joint Comput. Conf. (Dallas, 2—6 Nov. 1986). — Washington, 1986. — P. 41—47.
3. *Halbert D.C., O'Brian P.D.* Using types and inheritance in object-oriented programming // IEEE Software. — 1987. — 4, № 5. — P. 71—79.
4. *Wilson R.* Object-oriented languages reoriented programming techniques // Comput. Des. — 1987. — 26, № 20. — P. 52—62.
5. *Замулин А.В.* Типы данных в языках программирования и базах данных. — Новосибирск: Наука, 1987. — 149 с.
6. *Вирт Н.* МОДУЛА-2 // Языки программирования. — М.: Наука, 1985. — С. 3—46.
7. *Серебряков В.А.* Методы атрибутной трансляции // Там же. — С. 47—79.
8. *Дал У.* Языки для моделирования систем с дискретными событиями // Языки программирования. — М.: Мир, 1972. — С. 344—400.



Поступила 05.12.88
(после доработки — 04.04.89)


Тел. для справок: 32-99-79 (Куйбышев)


◆


УДК 681.3.06



**Ю.Б.Борзов, И.Э.Медведис, Г.Б.Уртанс**


# Метод сегментов для решения систем равенств и неравенств при генерации тестов для проверки программ

99

## Обзор методов решения условий реализуемости пути

В середине 70-х годов появились первые экспериментальные системы автоматического построения тестов (САПТ), использующие символьное выполнение программ [1]. САПТ обычно предназначаются для автоматического построения полной системы тестов относительно заданного структурного критерия тестирования [2] по тексту программы. Наличие символьного интерпретатора в САПТ позволяет получить условия реализуемости пути (УРП) программы — условия прохождения заданного пути программы [3]. УРП, как правило,

выражаются системой равенств и неравенств над начальными значениями входных переменных программы для выяснения реализуемости пути и построения теста (тестом будем называть набор значений входных переменных, при котором выполняется определенный путь программы).

В некоторых САПТ для решения УРП применяются традиционные методы (например, методы линейного программирования [4—7]), либо диалоговые методы [4]. Хотя имеется немало методов и пакетов программ для решения систем равенств и неравенств [6—8], применение существующих пакетов при построении тестов практически исключается по следующим причинам:

— разнотипность данных в языке программирования (числовые данные, строки символов и битов и т. д.);

— наличие в языках программирования специфических встроенных функций, нередко не имеющих однозначных инверсных функций;

— неопределенности, вносимые индексированными переменными (например, $a[i] = a[j]$ можно получить, либо положив $i = j$, либо присвоив $a[i]$ и $a[j]$ одно и то же значение), обращениями к подпрограммам и т. д.

Учитывая вышесказанное, авторы САПТ [9, 10] разработали собственные решатели УРП, которые характеризуются следующими особенностями: ведется поиск одного частного решения системы (частное решение является тестом для выполнения пути программы, заданного УРП); решатель находит решение не всегда — система, имеющая решение, может быть определена как не имеющая решения (вероятность такой ситуации увеличивается с возрастанием сложности УРП). Последнего, кстати, и следовало ожидать ввиду алгоритмической неразрешимости проблемы решения произвольных систем уравнений.

Если в других областях применения, кроме решения УРП, упомянутые свойства (особенно второе) могут считаться недопустимыми, то для САПТ они вполне приемлемы.

Ненахождение решения УРП может лишь снизить характеристику полноты построенной системы тестов, но не внесет существенных ошибок в результат — отрицательный ответ решателя повлечет за собой лишь вывод о нереализуемости определенного пути программы.

Исторически первым методом данного типа для решения УРП явился метод проб и ошибок, описанный в [9]. Кратко рассмотрим этот метод.

**Шаг 1.** Переведем УРП в конъюнктивную форму, т. е. $c_1 \,\&\, c_2 \,\&\, \ldots \&\, c_n$, где $c_i$ — неравенство или дизъюнкция.

**Шаг 2.** Упорядочим все переменные (неизвестные) и обозначим их $v_1, v_2, \ldots, v_m$.

**Шаг 3.** Разделим уравнения УРП на классы $s_1, s_2, \ldots, s_m$, такие, что уравнения из класса $s_i$ содержат переменную $v_i$ и, возможно, переменные $v_1, v_2, \ldots, v_{i-1}$.

**Шаг 4.** Последовательно определим значения всех переменных, начиная с $v_1$.

**Итерация** $i$. Подразумевается, что значения $v_1, \ldots, v_{i-1}$ уже определены и являются решением систем уравнений $s_1, \ldots, s_{i-1}$.

Значение $v_i$ определяется следующим образом.

1. Проверяем, содержит ли $s_i$ хоть одно равенство. Если да, то переходим к п. 2, если нет — к п. 3.

2. Решаем это равенство относительно $v_i$. Подставляем полученное значение $v_i$ в остальные уравнения $s_i$. Если полученное значение является решением всех уравнений $s_i$, переходим к итерации $i + 1$, если нет — возвращаемся к итерации $i - 1$ для генерации другого значения $v_{i-1}$.

3. Генерируем случайное значение $v_i$ в зависимости от его типа. Если полученное значение не является решением $s_i$, повторяем генерацию фиксированное число раз. Если все попытки заканчиваются безуспешно, то возвращаемся к итерации $i - 1$ для получения другого значения $v_{i-1}$ (или обращаемся за помощью к пользователю).

4. Если возвращение достигло итерации 1, то определяем УРП как не имеющие решения и заканчиваем работу.

Следует отметить несколько пожеланий при работе алгоритма, выполнение которых необходимо (но не всегда возможно) для его успешной работы:
– для решения проблемы неопределенности индексных переменных при упорядочении переменных (шаг 2) переменные-индексы следует ставить перед соответствующими переменными-массивами;
– переменные надо упорядочивать (шаг 2) так, чтобы уравнения из $s_i$ имели при генерации значений только одну неизвестную переменную $v_i$;
– способ генерации значения должен быть приспособлен к каждому виду уравнения.

Другой метод, относящийся к вышеупомянутому классу решателей УРП, — метод сегментов — был предложен для систем СМОТЛ [10] и в дальнейшем развит при разработке первой версии систем ТЕСТГЕН [11]. Несмотря на использование этого метода в двух системах тестирования, систематизированное изложение метода впервые предложено в настоящей работе.

## Основные положения метода сегментов

Можно считать, что метод сегментов представляет собой целенаправленное угадывание решения УРП. Решение проводится в три этапа.

1. *Установка начальных ограничений.* Устанавливаются начальные ограничения на область допустимых значений (ОДЗ) каждого элемента (т. е. переменной или выражения) системы в соответствии с определением переменной, ограничений, заданных пользователем, и т. п.

2. *Уточнение ограничений.* Последовательно уточняются (сокращаются) ОДЗ для каждого элемента системы с использованием закономерностей арифметических и логических операций, эвристик для обработки входящих в систему индексированных переменных (массивов), обращений к подпрограммам, встроенных функций и т. д.

3. *Выбор конкретных значений.* Конкретное значение теста выбирается из уточненной ОДЗ очередной рассматриваемой переменной системы случайным образом.

После этого опять уточняются ОДЗ и т. д. Если выясняется, что на некотором шаге решения ОДЗ какой-либо переменной или выражения система пуста или выбираемые конкретные значения все-таки не удовлетворяют систему (из-за примененных эвристик), делаем вывод об отсутствии решения. Таким образом, возможны четыре случая:
– система имеет решение и в результате получен конкретный тест (подчеркнем, что решение в общем виде не ищется);
– система на самом деле имеет решение, но получить тест не удалось;
– система не имеет решения, и в результате доказано отсутствие теста;
– система на самом деле не имеет решения, но доказательство отсутствия решения не получено.

С точки зрения построения теста все случаи, кроме первого, эквивалентны, так как приходится объявлять путь, соответствующий УРП, нереализуемым (возможно, ошибочно).

**Работа метода сегментов**

Покажем схему работы метода сегментов на простом примере. Пусть система неравенств

$$y \geq 3,$$
$$x \geq y + 2,$$
$$x + y \leq 20,$$

где $x$, $y$ — значения переменных типа BIN FIXED(15) языка ПЛ-1.

Данная система содержит простые элементы (переменные и константы): $x$, $y$, 2, 3, 20, выражения (арифметические и логические): $(x + y)$, $(y + 2)$ и все три уравнения системы. Рассмотрим работу всех этапов метода.

*Установка начальных ограничений.* Обозначим области допустимых значений функцией ОДЗ от элемента (переменной, выражения, константы). Допустимые области констант являются точками: ОДЗ(2) = [2, 2], ОДЗ(3) = [3, 3], ОДЗ(20) = [20, 20].

Начальные ОДЗ остальных элементов зададим на основе типа переменных, в которых эти значения хранятся. Как правило, этими областями являются сегменты [MIN, MAX], где MIN, MAX — константы, определяющие границы значений данного типа. Для BIN FIXED(15) в ПЛ-1 MIN = − 32767, MAX = 32767.

Таким образом для данного примера устанавливаются следующие начальные ограничения: ОДЗ($x$) = ОДЗ($y$) = ОДЗ($x + 2$) = ОДЗ($y + 2$) = [MIN, MAX].

*Уточнение ограничений.* Используем правила уточнения (минимизации) для арифметических операций и операций сравнения, описанных ниже. Результаты всех уточнений этапа показаны в табл. 1. Прокомментируем некоторые из них. Сначала рассмотрим пример уточнений при операциях сравнения: уточнение 1 из табл. 1. Неравенство $y \geq 3$ над операндами $y$ (ОДЗ($y$) = [MIN, MAX]) и 3 (ОДЗ(3) = [3, 3]). Правило уточнения для операции "$\leq$" основывается на том, что ОДЗ операнда левой стороны неравенства ($y$) не должна содержать значения, меньшие минимально допустимого значения правой стороны (константы 3). Правило для ОДЗ правого операнда аналогично.

После применения этого правила получаем ОДЗ($y$) = [3, MAX], ОДЗ(3) не изменяется.

Рассмотрим пример уточнений при арифметических операциях: уточнение 5 выражения $x + y$. Перед уточнением ОДЗ($x + y$) = [MIN, 20], ОДЗ($x$) = [MIN, MAX], ОДЗ($y$) = [3, MAX]. Из монотонности операции сложения следует, что максимально допустимое значение суммы не может превышать суммы максимальных значений слагаемых, а минимальное значение суммы не может быть меньше суммы минимальных значений слагаемых. С помощью несложных преобразований можно получить похожие уточнения и для слагаемых. Таким образом, после уточнения получаем: ОДЗ($x + y$) = [MIN, 20], ОДЗ($x$) = [MIN, 17], ОДЗ($y$) = [3, MAX].

Этап уточнения заканчивается, если при очередном просмотре всех выражений не получено новых ограничений (см. уточнения 11—15 в табл. 1).

*Выбор конкретных значений.* После уточнения ограничений генерируем поочередно значения переменных из их ОДЗ. Предположим, что генератор случайных чисел из ОДЗ($x$) = [5, 17] в качестве значения $x$ выбрал 13 (так как $x$ — значение типа BIN FIXED(15), генерируется целочисленное значение).

Таблица 1. Пример уточнения ограничений

| N п/п | Выражение | ОДЗ перед уточнением | | | | ОДЗ после уточнения | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $x$ | $y$ | $x+y$ | $y+2$ | $x$ | $y$ | $x+y$ | $y+2$ |
| 1 | $y \geq 3$ | - | [MIN,MAX] | [MIN,MAX] | - | - | [3,MAX] | - | - |
| 2 | $x \geq y+2$ | [MIN,MAX] | - | - | [MIN,MAX] | [MIN,MAX] | - | - | [MIN,MAX] |
| 3 | $x+y \leq 20$ | - | - | [MIN,MAX] | - | - | - | [MIN,20] | - |
| 4 | $y+2$ | - | [3,MAX] | - | [MIN,MAX] | - | [3,MAX] | - | [5,MAX] |
| 5 | $x+y$ | [MIN,MAX] | [3,MAX] | [MIN,20] | - | [MIN,17] | [3,MAX] | [MIN,20] | - |
| 6 | $y \geq 3$ | - | [3,MAX] | - | - | - | [3,MAX] | - | - |
| 7 | $x \geq y+2$ | [MIN,7] | - | - | [5,MAX] | [5,17] | - | - | [5,17] |
| 8 | $x+y \leq 20$ | - | - | [MIN,20] | - | - | - | [MIN,20] | - |
| 9 | $y+2$ | - | [3,MAX] | - | [5,17] | - | [3,15] | - | [5,17] |
| 10 | $x+y$ | [5,17] | [3,15] | [MIN,20] | - | [5,17] | [3,15] | [8,20] | - |
| 11 | $y \geq 3$ | - | [3,15] | - | - | - | [3,15] | - | - |
| 12 | $x \geq y+2$ | [5,17] | - | - | [5,17] | [5,17] | - | - | [5,17] |
| 13 | $x+y \leq 20$ | - | - | [8,20] | - | - | - | [8,20] | - |
| 14 | $y+2$ | - | [3,15] | - | [5,17] | - | [3,15] | - | [5,17] |
| 15 | $x+y$ | [5,17] | [3,15] | [8,20] | - | [5,17] | [3,15] | [8,20] | - |
| Конечное состояние ОДЗ | | | | | | [5,17] | [3,15] | [8,20] | [5,17] |

Проведем дополнительное уточнение, используя полученное значение $x = 13$, т. е. ОДЗ($x$) = [13, 13]. После уточнения получаем ОДЗ($y$) = [3, 7]. Предположим, что генератор случайных чисел из [3, 7] выбрал 5. Проверяем, является ли пара $x = 15$, $y = 5$ решением, и, убедившись в этом, заканчиваем работу успешно.

Вернемся к началу угадывания. Предположим, что в качестве первого угадывания было выбрано не $x$, а $y$ (ОДЗ ($y$) = [3, 15]), и генератор случайных чисел выбрал значение $y = 12$. В этом случае после уточнения получаем противоречие $x \leq 8$, $x \geq 14$, и попытка угадывания заканчивается безуспешно. Можем сделать еще несколько попыток (число их может зависеть от временных ресурсов, предоставленных для решения) и в случае безуспешного завершения их всех возвращаем ответ "Система слишком сложна для решения" или "Решения нет".

Перейдем к более общему рассмотрению этапов метода сегментов.

**Установка начальных ограничений.** При решении УРП каждому элементу будет соответствовать область допустимых значений. Начальные ОДЗ формируются на этапе установки начальных ограничений исходя из описаний переменных в тексте программы, форматов ввода-вывода данных (если значения считываются с файла), указаний программиста (в тексте программы, спецификации или в диалоге с решателем), некоторых эвристических предположений, заложенных в решатель.

Первых два ограничения являются объективными ограничениями на ОДЗ, за пределы которых значения переменных и выражений не могут выйти в принципе. Остальные ограничения являются субъективными предположениями программиста и автора решателя о вероятнейших областях значений переменных и выражений. Начальные ОДЗ будем представлять в виде сегментов (отсюда название метода) и формировать следующим образом.

*Определение элементов.* Константе $c1$ присвоим сегмент $[c1, c1]$; для строковых констант (символьных или битовых) сегмент (точка) присваивается каждому элементу строки. Переменной присвоим сегмент, ограниченный экстре-

**Таблица 2. Конечные ОДЗ для некоторых операций сравнения**

| Операция | Первый операнд | Второй операнд |
|---|---|---|
| ≤ | [*a*1,min(*a*2,*b*2)] | [max(*a*1,*b*1),*b*2] |
| = | [max(*a*1,*b*1),min(*a*2,*b*2)] | [max(*a*1,*b*1),min(*a*2,*b*2)] |

мальными значениями данного типа (например, [– 32767, 32767] для BIN FIXED(15), '0'B, '1'B для BIT(1)); для строковых переменных интервал присваивается каждому элементу строки.

*Форматы ввода-вывода.* Если переменные получены при чтении файла, ОДЗ ограничиваются экстремальными значениями, допускающими ввод при заданном формате. Так, формат в ПЛ-1 допускает ввод чисел [– 99, 999].

*Указания программиста.* Программист может сам указать необходимые ограничения для переменных, как частный случай может использовать фиксацию каких-либо переменных. Указания могут быть заданы как в тексте или спецификации программы, так и в диалоге с решателем.

*Эвристики, заложенные в решатель.* На практике оказался полезным также один весьма грубый эвристический прием — нахождение в тексте наибольшей по абсолютной величине константы $a$ заданного типа и дополнение ограничений переменных этого типа сегментом $[-20-a, 20+a]$.

В последних двух случаях нужно быть осторожным, так как неправильное предположение может отрицательно повлиять на результат. Если эвристики не окажутся неоспоримыми, следует их применение перенести на дальнейшие этапы решения, где отрицательные последствия менее значительны.

**Уточнение ограничений.** Задача второго этапа метода сегментов — минимизация (сужение) ОДЗ всех элементов УРП (исходя из ОДЗ, полученных на первом этапе) с тем, чтобы увеличить вероятность быстрого угадывания решения в дальнейшем. При этом на основе анализа УРП минимизируются ОДЗ не только для переменных, но и для выражений.

Алгоритм сужения ОДЗ заключается в последовательном просмотре выражений и попытке минимизации ОДЗ согласно заданным правилам минимизации, которые зависят от операции данного выражения, типов операндов и видов их ОДЗ (интервалы, сегменты, объединение сегментов и т. д.).

Алгоритм заканчивается, если после очередного просмотра всех выражений вершин не сузилась ни одна ОДЗ. В принципе можно прекратить сужение в любой момент, но тогда исходные данные для шага угадывания будут менее качественными.

Объем статьи не позволяет привести обширный набор правил минимизации, поэтому ограничимся несколькими примерами. Возьмем исходные ОДЗ: сегменты [*a*1, *a*2] — для первого операнда; [*b*1, *b*2] — для второго; [*c*1, *c*2] — для результата обозреваемой операции. Покажем конечные ОД для некоторых операций сравнения (ОДЗ результата в этом случае будем считать ['1'B, '1'B] (табл. 2)).

Правила минимизации для арифметических операций основаны на их монотонности. Правила для основных операций (ОДЗ операций и операндов — это ранее описанные сегменты) приведены в табл. 3.

Упомянутые правила для "+" и "–" действуют для всей области значений, а "*" и "/" — для сегментов с положительными границами. Аналогичные правила можно написать для этих операций и для других областей. Тогда необходимо разделить исходные ОДЗ на несколько сегментов и полусегментов, а затем применить для каждого соответствующее правило. В этом случае

исходные ОДЗ уже не являются сегментами, а объединением сегментов (полусегментов, интервалов). При работе с такими объектами соответствующие правила применяются для всех комбинаций (троек) сегментов, и результаты всех минимизаций соответствующего элемента объединяются.

Таблица 3. Конечные ОДЗ для основных операций

| Операция | Первый операнд | Второй операнд | Операция |
|---|---|---|---|
| + | [max(a1,c1-b2), min(a2,c2-b1)] | [max(b1,c1-a2), min(b2,c2-a1)] | [max(c1,a1+b1), min(c2,a2+b2)] |
| - | [max(a1,b1+c1), min(a2,b2+c2)] | [max(b1,a1-c2), min(b1,a2-c1)] | [max(c1,a1-b2), min(c2,a2-b1)] |
| * | [max(a1,c1/b2), min(a2,c2/b1)] | [max(b1,c1/a2), min(b2,c2/a1)] | [max(c1,a1*b1), min(c2,a2*b2)] |
| / | [max(a1,b1*c1), min(a2,b2*c2)] | [max(b1,a1/c2), min(b1,a2/c1)] | [max(c1,a1/b2), min(c2,a2/b1)] |

Таким образом, новой ОДЗ элемента становится объединение сегментов. Помимо арифметических операций и операций сравнения выражения могут содержать также обращения к встроенным функциям. Дадим пример правил минимизации для обращений к встроенным функциям. Пусть исходная ОЗ аргумента $x[a1, a2]$, а выражения функции $ABS(x)$ — $[c1, 2]$. Тогда после уточнения:

ОДЗ($ABS(x)$) = [max(c1), min($ABS(a1)$, $ABS(a)$), min(c2), max($ABS(a1)$, $ABS(a2)$)];
ОДЗ($x$) = [max($a1$, $-c2$), min($a2$, $-c1$)] ∪ [max($a1$, $-c1$), min($a2$, $c2$)].

При применении правил минимизации для работы с объединением сегментов могут появиться противоречивые (пустые) сегменты (например, [3, 1]), их следует просто опускать. Однако, если после минимизации вся ОДЗ какого-либо элемента оказывается пустой (противоречивой), то все УРП противоречивы (решения нет).

Число сегментов в ОДЗ при повторных уточнениях может возрастать. Для того чтобы число сегментов в ОДЗ элемента не превышало допустимого при реализации, можно использовать эвристику — объединение некоторых сегментов ОДЗ с добавлением к ОДЗ интервала между ними и увеличением ОДЗ. Однако при этом появляется опасность зацикливания алгоритма и качество результата снижается.

Отметим также, что в случае фиксации значений всех переменных (на третьем этапе метода) алгоритм минимизации будет проверкой того, является ли данная фиксация решением.

Правила минимизации могут быть достаточно разнообразными. Наряду с объективными правилами, описанными выше, можно использовать и эвристические предположения. Возможны также фиксирования (угадывания) значений некоторых операндов и/или значений выражений уже на этапе уточнения. Такие действия могут как упростить (а значит, ускорить) решение УРП, так и существенно повлиять на нахождение результата, поскольку фиксация происходит лишь однажды — на этапе выбора конкретных значений предусмотрены возвраты при неудачном выборе значений. Поэтому на этом этапе те конструкции, для которых не разработаны (или пока не реализованы) правила уточнения, можно не уточнять.

**Выбор конкретных значений.** После получения уточненных ОДЗ переменных и выражений начинается "угадывание решения". Для этого используем схему выбора значений с возвратами, похожую на схему метода проб и ошибок,

кратко описанную при обзоре методов решения УРП. Существенное отличие между ними состоит в следующем:

— выбор конкретных значений переменных в нашем случае происходит только из ОДЗ, подготовленных предыдущими этапами метода, что увеличивает вероятность угадывания;

— после выбора очередного конкретного значения происходит дополнительное уточнение ОДЗ еще не фиксированных переменных (с помощью алгоритма уточнений второго этапа). Это позволяет увеличивать вероятность угадывания в дальнейшем.

Для выбора значений используется генератор псевдослучайных чисел. Для управления работой генератора предложим следующую эвристику. При повторном выборе значений очередной переменной (при неудачном выборе в предыдущих попытках) выполняется такая последовательность.

Пусть $c1$, $c2$, $c3$ — параметры решателя. Тогда выбирается $c1$ значений экстремальных точек сегментов, интервалов и полусегментов ОДЗ; выбирается $c2$ значений, близких к экстремальным точкам сегментов, интервалов и полусегментов ОДЗ; выбирается $c3$ значений из всей ОДЗ. Если за $c1 + c2 + c3$ попыток не удается получить необходимое значение переменной, происходит возврат к предыдущей итерации для получения другого значения.

Если несколько попыток выбора не дают положительного результата, можно подключить к решению пользователя для выбора (фиксации) значений некоторых переменных. В этом случае решатель превращается в интерактивного ассистента программиста для нахождения тестовых данных.

### Апробация метода сегментов

Применимость метода сегментов подвергалась испытанию дважды. Впервые он был запрограммирован для системы СМОТЛ [10]. Были получены положительные результаты построения тестов для программ, не превышающих 300 операторов СМОД [12]. Однако конкретный вид системы неравенств и их ход решения тогда не исследовались.

При разработке алгоритмов построения тестов для программ типа системного программирования, записываемых на подмножестве языка ПЛ-1 [13], после введения ряда специальных эвристик было решено предварительно провести ручной эксперимент с использованием таблиц псевдослучайных чисел.

Метод сегментирования был проверен на 21 программе, большинство из которых — типичные примеры системного программирования (внутренняя сортировка, слияние файлов, выделение оперативной памяти, сборка "мусора", поиск в таблице, обработка очереди заданий, форматизация текста, перекодировка представления даты, обход бинарного дерева и т. п.). В основном анализировались опубликованные программы и алгоритмы [14—19], а также несколько наиболее часто используемых подпрограмм самой экспериментальной реализации предложенных алгоритмов построения тестов. Размер программ изменялся от 7 до 76 выполняемых операндов (в среднем 22,9), мера сложности по Маккейбу — от 4 до 16 (в среднем 6,3).

В общей сложности было построено 56 тестов. Таким образом, не считая промежуточного решения УРП для выяснения реализуемости строящихся путей, следовало решить 56 УРП. Число неравенств в УРП изменялось от 1 до 17 (в среднем 4,1), число переменных — от 1 до 25 (в среднем 4,6).

Для получения визуального представления о решаемых системах (которые, конечно, зависят от выбранных для проверки методов программ) приведем такую, достаточно типичную систему и результат ее решения:

| *Система* | *Решение* |
|---|---|
| ∧(setno <0 \| setno> m(2)) | setno = 15 |
| ∧(m)setno + 4) = 0) | m(2) = 0 |
| m(m(5 + setno) + 4 + flag)∧ = 0 | flag = - 19 |
| lelem = 1 | m(1) = 22 |
| flag + 1 = m(m(setno + 5) +1) | m(3) = 27 |
| m(m(5 + setno) + 5 + flag) = 0 | m(20) = 16 |
| m(m(5 + setno) + 6 + flag)∧ = 0 | lelem = 1 |

Предложенный алгоритм позволил решить все системы, хотя в восьми случаях была достаточно большая вероятность фиксировать значения, не позволяющие получить решения.

Вышесказанное дает достаточно серьезное основание считать метод сегментов перспективным для использования в автоматическом построении тестов (конечно, для каждого конкретного применения он должен дополняться конкретными эвристиками).

В настоящее время метод перерабатывается для применения к внутреннему языку HTEXT отладочного компилятора ПЛ-1 с целью обеспечения автоматизации построения тестов для программ на языке ПЛ-1.


1. *Борзов Ю.В.* Тестирование программ с использованием символического выполнения // Программирование. — 1980. — № 1. — С. 51—59.
2. *Борзов Ю.В., Уртанс Г.Б., Шимаров В.А.* Выбор путей программы для построения тестов // УСиМ. — 1989. — № 6. — С. 29—36.
3. *Уртанс Г.Б.* Функции путей программ // Программирование. — 1987. — № 4. — С. 69—78.
4. *Bayer R.S., Elspas B., Levitt K.M.* SELECT — a formal system for testing and debugging programs by symbolic execution // Proc. Intern. Conf. on Reliable Software. — Los Angeles, 1975. — P. 234—245.
5. *Clarke L.A.* A system to generate test data and symbolically execute programs // IEEE Trans. on Software Eng. — 1976. — 2, № 3. — P. 215—222.
6. *Bendars J.F.* Partitioning procedures for solving mixed-variables programming problems // Numerische Mathematik. — 1962. — № 4. — P. 238—252.
7. *Gomory R.E.* An algorithm for integer solution to linear programs // Recent Advances in Mat. Programming. — N.Y. : McGraw-Hill, 1963. — P. 130—210.
8. *Derman E., Van Wyk C.* A simple equation solver and its application to financial modelling // Software — Practice and Experience. — 1984. — № 14. — P. 1169—1181.
9. *Ramamoorthy C.V., Ho Siu-Bun F., Chen W.T.* On the automated of programm test dete // IEEE Trans. on Software Eng. — 1976. — 2, № 4. — P. 293—300.
10. *SMOTL* — a system to construct cimples for data processing programm debuggin / J. Bicevskis, J.Borzovs, U.Straujums, A.Zarins, .F.Viller(Jr)// Ibid. — 1979. — 5, № 1. — P. 60—66.
11. *Система* символического тестирования ПЛ-1-программ / Ю.В.Борзов, Г.Э.Дишлес, И.Э.Медведис, Г.Б.Уртанс // УСиМ. — 1986. — № 6. — С. 75—79.
12. *Инструкция* по работе с системой макрокоманд обработки данных (СМОД). Ч. 1. Основные операторы / Я.Я.Бичевский, А.К.Зариньш, А.А.Калиньш и др. // Ассоциация пользователей ЭВМ типа "Минск". — 1973. — Вып. 9—10. — С. 1—109.
13. *Катков Б.Л., Пилецкий И.И.* Подмножество языка ПЛ-1 для системного программирования // УСиМ. — 1983. — № 3. — С. 59—63.
14. *Берзтис А.Т.* Структуры данных. — М.: Статистика, 1974. — 408 с.
15. Вирт Н. Системное программирование. Введение. — М.: Мир, 1976. — 183 с.
16. *Кнут Д.* Искусство программирования для ЭВМ. Основные алгоритмы. — М.: Мир, 1976. — 735 с.
17. *Hoare C.A.R.* Algorithms 65; FIND // Comm. ACM. — 1961. — 4, № 1. — P. 321.

18. *Kernighen B.W., Plauger P.J.* The elements of programming style. — N.Y.: McGraw-Hill, 1974. — 168 p.
19. *Naur P.* Programming by action clusters // BIT. — 1969. — 9, № 3. — P. 250—258.
20. *McCabe T.J.* A complexity measure // IEEE Trans. Software Eng. — 1976. — 2, № 4. — P. 308.

Поступила 14.12.88.
Тел. для справок: 22-54-97 (Рига)

◆

УДК 519.95

**С.В.Игнатов**

# Язык спецификации математических моделей систем с дискретной составляющей состояния (краткое сообщение)

В модельно-ориентированном пакете прикладных программ "Анализ процессов эксплуатации сложных технических систем" для решения задач оценки показателей надежности и оптимизации технического обслуживания используется метод имитационного моделирования [1]. Ориентация пакета на конечного пользователя потребовала реализации в нем процедур автоматического синтеза программ моделирования. Схема синтеза представляет собой последовательность преобразований информационной модели объекта исследования и постановки задачи сначала в математическую, а затем в алгоритмическую модели [2]. Если для информационной и алгоритмической моделей использованы традиционные способы их задания, а именно предикаты и универсальный алгоритмический язык, то для спецификации математической модели был разработан специальный язык декларативного типа.

Данный язык построен на базе понятия кусочно-непрерывной агрегативной системы [3] и называется КНА1-языком.

Описание математической модели на КНА1-языке включает спецификации автономного поведения агрегатов и их взаимодействия. Тот факт, что в анализируемых системах элементы часто формализуются с помощью одних и тех же математических отношений, позволяет сократить описание за счет введения класса агрегатов. В КНА1-языке допускается два типа агрегатов: с конечным (К) и счетным (С) множеством основных состояний. Задание класса агрегатов типа К состоит из спецификации всех его основных состояний и имеет вид

КЛ_АГРЕГАТА <имя> (К); <список описаний основных состояний>;
КОН_КЛ_АГРЕГАТА;

Объявление одного основного состояния выполняется с помощью оператора вида

ОСН_СОСТ <номер>;
<список описаний дополнительных координат>;
<список описаний переходов>;
КОН_ОСН_СОСТ;

Описание основного состояния включает описание изменения дополнительных координат в данном состоянии и описание переходов из этого состояния в другие. Описание дополнительной координаты следующее:

КООРД <идентификатор дополнительной координаты>, <тип координаты>
[, <вид зависимости> (<список координат, являющихся параметрами>)];

Тип координат указывает на характер ее зависимости от времени, других координат и параметров агрегата. Вид зависимости — это имя функции, по которой рассчитывается значение координаты и определяется время до пересечения координатой заданной границы. Библиотека функций является открытой. Она пополняется или настраивается для конкретных классов исследуемых систем. Далее в скобках перечисляются имена координат, которые являются параметрами указанной функции.

Оператор объявления перехода задается конструкцией вида

Таким образом, даже при несовершенном аппаратном контроле вычислительных средств можно предъявлять достаточно высокие требования к надежности функционирования реализуемых на этих вычислительных средствах программ.

1. *Халецкий А. К.* Методика оценки надежности функционирования управляющей программы // Надежность и контроль качества.— 1981.— № 4.— С. 12—18.
2. *Шаракшанэ А. С., Шахин В. П., Халецкий А. К.* Испытания программ сложных автоматизированных систем.— М. : Высш. шк., 1982.— 192 с.
3. *Липаев В. В.* Проектирование математического обеспечения АСУ : Системотехника, архитектура, технология.— М. : Сов. радио, 1977.— 400 с.
4. *Средства* отладки больших систем / Под ред. Р. Растина.— М. : Статистика, 1977.— 136 с.

Поступила 06.05.85
(после доработки — 13.01.86)

УДК 681.3.06

# Система символического тестирования ПЛ/1-программ

**Ю. В. Борзов, Г. Э. Дишлерс, И. Э. Медведис, Г. Б. Уртанс**

**Назначение системы.** Система символического тестирования ПЛ/1-программ ТЕСТГЕН является экспериментальной подсистемой инструментального комплекса системного программирования. Ее основная функция — автоматическая генерация полных систем тестов [1, 2], т. е. таких наборов тестов, при которых проверяются все реализуемые выходы операторов *. Помимо этого, предваряющий генерацию статический анализ программы позволяет выявить недостижимые и существенные переменные [4—7], а также характеристики сложности, например, нормальное число программ [8]. Для выбранных путей программы строятся функции путей, т. е. отображения входных данных в выходные, задаваемые этими путями.

Входным языком системы является неточное подмножество ПЛ/1, описанное в [9]. Оно характеризуется наличием только целочисленной двоичной арифметики, строковыми данными сильно ограниченной длины, жесткими правилами использования оператора цикла, декларацией всех переменных, отсутствием ввода-вывода и рядом менее существенных ограничений. Добавлены возможности фильтрации значений переменных [10].

Поскольку основные алгоритмы символического тестирования неизбежно носят эвристический характер, авторы задались целью создания действующей системы, не заботясь об ее оптимальности. ТЕСТГЕН почти полностью написана на входном языке самой системы с использованием специально созданных подпрограмм, реализующих наиболее распространенные операции символического тестирования. Предполагается в ходе опытной эксплуатации выявить те части системы, которые отрицательно влияют на оптимальность как по времени, так и по памяти, и принять меры для их оптимизации.

**Пример работы системы.** В последующих статьях предполагается подробно описать стратегию выбора путей программы для построения тестов, символическое выполнение путей, решение условий реализуемости путей и привести статистику работы системы. Здесь же мы просто продемонстрируем основные фазы работы ТЕСТГЕН на простейшей ПЛ/1-программе, которая в действительности служила в качестве самого первого комплексного теста системы:

```
TEST1: PROCEDURE (X, Y);                    1
  DECLARE (X, Y) BINARY FIXED (15);         2
  IF X<17 THEN                              3
        Y=1;                                4
  IF X>=17 THEN                             5
        Y=2;                                6
  END TEST1;                                7
```

В итоге эксплуатации ТЕСТГЕН получены следующие результаты:

— полная система тестов:

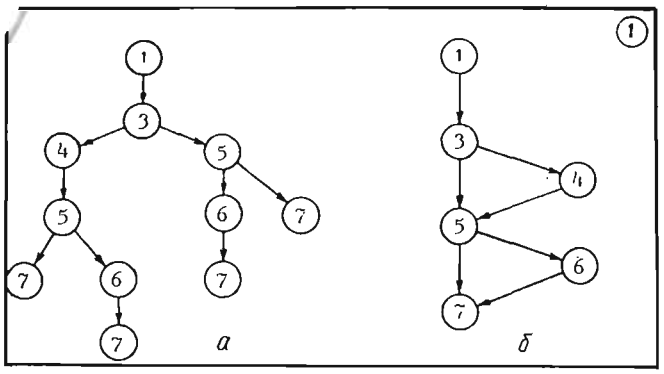
| Тест 1 | Тест 2 |
|---|---|
| X=—24 | X=19 |
| Y=0 | Y=47; |

— система функций покрывающего множества путей:

| Функция пути (1, 3, 4, 5, 7) | Функция пути (1, 3, 5, 6, 7) |
|---|---|
| X=X | X=X |
| Y=1 | Y=2 |
| X<17 | ¬ (X=17) |
| ¬ (X>=17) | X>=17; |

— неинициализированная переменная — X;
— существенная переменная — X;
— неиспользуемые переменные — нет;
— недостижимые операторы — нет;
— существенные операторы — 1, 7;
— нормальное число программы — 0.

* Известно [3], что автоматическое построение полной системы тестов является алгоритмически неразрешимой проблемой для реальных языков программирования. Поэтому авторы предлагают один эвристический алгоритм, который, по их замыслу, должен выполнять поставленную задачу в большинстве практических случаев.




**Внутреннее представление тестируемой программы.** Символическое тестирование обычно требует многократного просмотра текста тестируемой программмы в различных ракурсах. Входной язык системы неудобен для этой цели и поэтому перед началом анализа препроцессор переводит синтаксически правильную тестируемую программу во внутреннее представление. Язык внутреннего представления по уровню максимально приближен к ПЛ/1, но имеет следующие основные особенности:

— обеспечена перемещаемость внутреннего представления тестируемой программы путем адресации всех объектов относительно начала некоторого выделенного поля памяти;

— широко используются таблицы для хранения информации о переменных, константах, метках, файлах и т. д. Операнды выполнимых операторов представляют собой ссылки на разделы соответствующих таблиц;

— последовательности операторов внутреннего представления, являющиеся результатом трансляции одного оператора ПЛ/1 тестируемой программы, «обрамляются» специальными операторами НАЧАЛО-ВНЕШНЕГО-ОПЕРАТОРА и КОНЕЦ-ВНЕШНЕГО-ОПЕРАТОРА. Эти операторы содержат номер внешнего оператора и относительные адреса предыдущего и последующего(щих) внешних операторов как в смысле их естественного расположения в программе, так и в смысле передачи управления (точнее, указываются адреса соответствующих операторов НАЧАЛО-ВНЕШНЕГО ОПЕРАТОРА);

— оператор присваивания содержит не более двух операндов, и они должны быть однотипными. Всякая перекодировка типа и точности представления значения переменной допускается только в момент присваивания вычисленного значения выражения.

Программа TEST1 во внутреннем представлении с учетом неиспользуемой памяти (она выделяется блоками) занимает 680 байт, а вообще данная редакция системы может располагать в своей памяти ПЛ/1-программы длиной около 400 операторов.

**Статический анализ.** Перед началом статического анализа конструируется развертка программы в виде дерева, вершинами которого служат операторы тестируемой программы, а дугами — возможные передачи управления. На рис. 1, *а* приведена развертка программы TEST1. Поскольку при наличии цикла в программе такое дерево может оказаться бесконечным, проводится обрезание ветви, если на ней повторяется один и тот же номер оператора. Развертка программы служит удобным средством выявления ряда статических характеристик программы. Например, число ветвей, обрезанных по повторению номеров операторов, является одной из мер сложности — нормальным числом программы.

Недостижимым назовем оператор, к которому не ведет никакой путь в программе. Ясно, что все достижимые операторы содержатся в развертке программы. Разность между множеством всех выполнимых операторов программы и множеством операторов, содержащихся в развертке, представляет множество недостижимых операторов.

Неиспользуемыми назовем переменные, не входящие ни в какой выполняемый оператор ни на каком пути программы (возможны и более тонкие определения). Разность между множеством всех переменных программы и множеством тех переменных, которые входят в операторы, содержащиеся в развертке программы, представляет множество неиспользуемых переменных.

Неинициализированной называется переменная, значение которой используется в каком-либо операторе хотя бы на одном пути программы без предварительного присвоения значения переменной. Для обнаружения неинициализированных переменных надо просмотреть все ветви развертки программы в направлении от их концов к корню дерева (к первой выполняемой команде). Просматривая текущий оператор, все указанные в нем используемые переменные (например, входящие в правую часть оператора присваивания) заносим в множество «подозреваемых» переменных. Те переменные, которым в данном операторе значение присваивается (например, входящие в левую часть оператора присваивания), из множества подозреваемых удаляем. Таким образом, дойдя до корня дерева в множестве подозреваемых переменных, получим переменные, которым в данной программе не присвоено значение перед использованием хотя бы на одном пути (в частности, такими переменными должны оказаться входные параметры). Пройдя по всем ветвям развертки и объединив все множества подозреваемых переменных, получим множество неинициализированных переменных для всей программы.

Про существенные операторы и переменные с точки зрения пользователя можно сказать следующее: это те точки программы, в которые полезно вставлять отладочную выдачу промежуточных результатов, и та информация, которую следует выдавать в этих точках, так как она определяет дальнейшие передачи управления.

Существенными операторами назовем операто-

ры организации циклов в программах. Дополнительно первый выполнимый оператор программы и все операторы выхода из нее отнесем к существенным. Ясно, что можно считать все операторы программы существенными, но речь идет о минимизации множества.

Переменная называется существенной для данного оператора, если имеется путь, начинающийся с данного оператора, и такой, что справедливо следующее утверждение: значение переменной перед выполнением оператора анализируется в условном операторе этого пути и не меняется до момента его анализа в том же условном операторе пути.

Выявление существенных переменных и операторов более громоздко и здесь приводиться не будет.

**Генерация тестируемых путей.** Чаще всего в системах автоматизированного тестирования используются статические методы генерации тестируемых путей [11—14]. Пути для тестирования выбираются перед построением тестов, исходя из управляющего графа программы. При этом для достижения полного покрытия графа могут использоваться хорошо разработанные методы теории графов. К сожалению, выбранные таким образом пути могут потом оказаться нереализуемыми и тогда придется или повторять выбор путей, или довольствоваться неполным покрытием.

Используемый нами метод генерации путей назовем динамическим в противоположность статическим методам. Уже в момент выбора пути для тестирования используется информация о реализуемости, поэтому не приходится блуждать по управляющему графу программы «вслепую».

Назовем линейные участки программы линейными путями (ЛП). Последовательность ЛП, ведущую от начала программы, назовем путем; путь, достигающий конца программы — стоп-путем. Далее приведены основные правила, которыми мы руководствуемся при генерации путей.

Во-первых, генерируются только реализуемые пути и сгенерированным считается такой путь, который достигает выхода из программы. Во-вторых, генерация путей происходит постепенно, к одному ЛП присоединяется другой, причем только такой ЛП, что весь сгенерированный путь реализуем. В-третьих, мы пытаемся всегда присоединять такие ЛП, которые еще не включены в генерируемый или ранее сгенерированные пути. В-четвертых, при выборе очередного присоединяемого ЛП принимается во внимание семантика последнего оператора ЛП. Затем делается попытка определить естественное продолжение генерируемого пути. Если это оператор ввода, то, очевидно, естественно пойти по пути ввода информации и только во второй раз, попадая на этот же ЛП, выбрать выход на состояние ENDFILE. Если это оператор заголовка цикла, то надо пытаться попасть в тело цикла. Если это оператор IF, то необходимо выбрать сначала выход по THEN.

Для рассматриваемой программы управляющий граф приведен на рис. 1, *б*.

Продемонстрируем процесс генерации путей.

Шаг 1. Исходным ЛП выбираем $ЛП_1 = (1, 3)$.

Шаг 2. Имеется два варианта продолжения полученного пути (1, 3): $(ЛП_1, ЛП_2) = (1, 3, 4, 5)$ и $(ЛП_1, ЛП_3) = (1, 3, 5)$. Проверяем их реализуемость, обращаясь к символьному интерпретатору и решателю неравенств. В данном случае оба пути реализуемы. Тогда выбираем путь $(ЛП_1, ЛП_3)$. Поскольку оператор 3 исполнялся впервые, отмечаем это и запоминаем, что мы могли выбрать выход (3,5) и продолжать генерацию по пути $(ЛП_1, ЛП_3)$; запоминаем, что выход (3, 5) пока не реализован.

Шаг 3. Имеем путь $(ЛП_1, ЛП_2) = (1, 3, 4, 5)$. Присоединяем два следующих «кандидата» на продолжение полученного пути. Получаем два пути: $(ЛП_1, ЛП_2, ЛП_4)$ и $(ЛП_1, ЛП_2, ЛП_5)$. Проверяем их реализуемость: реализуем только путь $(ЛП_1, ЛП_2, ЛП_5)$ Поскольку оператор 5 исполнялся впервые, отмечаем это и запоминаем, что выход (5, 6) пока не реализован.

Шаг 4. Поскольку последний оператор в $ЛП_5$ является оператором конца программы, то сгенерирован реализуемый от начала до конца программы путь $(ЛП_1, ЛП_2, ЛП_5) = (1, 3, 4, 5, 7)$. Обобщим информацию, связанную с этим стоп-путем: от него имеется два нереализованных ответвления с выходами (3, 5) и (5, 6), причем ответвление (3,5) в принципе реализуемо.

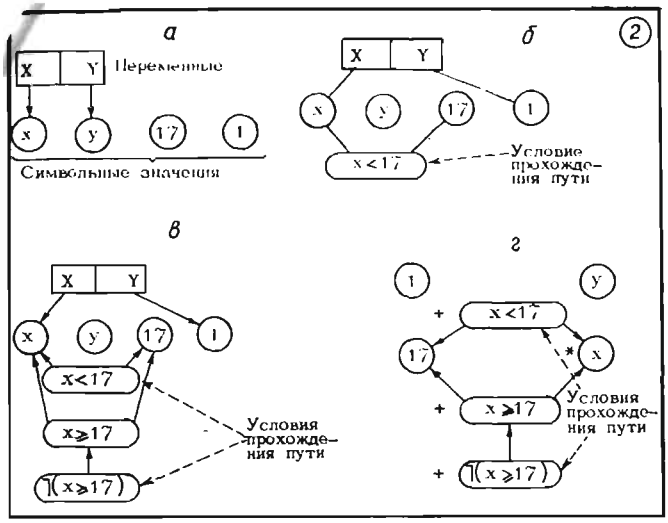
Шаг 5. Теперь необходимо выбрать следующий путь. Основой выбора начала следующего пути является информация о реализуемых выходах. В данном примере есть только один выход, который пока не реализован, но в принципе реализуем. В общем случае могут быть несколько таких «кандидатов» на возможное продолжение нового генерируемого пути. Пока в описываемой системе заложен принцип: выбрать путь, ближайший к началу программы. Итак, выбираем в качестве продолжения $ЛП_3$ (другого варианта просто нет). Поскольку оператор 3 проходится повторно (первый проход отмечен при выполнении шага 2), проверяем, не отмечен ли выход (3,5) как пока не реализованный. Уничтожаем этот признак.

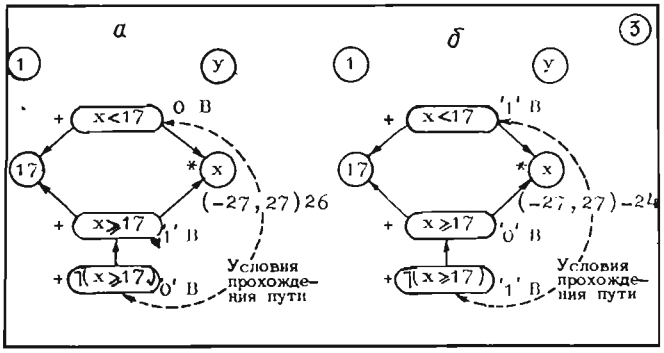
Шаг 6. Имеем путь $(ЛП_1, ЛП_3)$, на продолжение которого претендуют $ЛП_4$ и $ЛП_5$. После проверки реализуемости оказывается, что реализуем только путь $(ЛП_1, ЛП_3, ЛП_4)$. Поскольку оператор 5 проходится вторично, то для выхода (5,6) установленный на шаге 3 признак «пока не реализован» уничтожается.

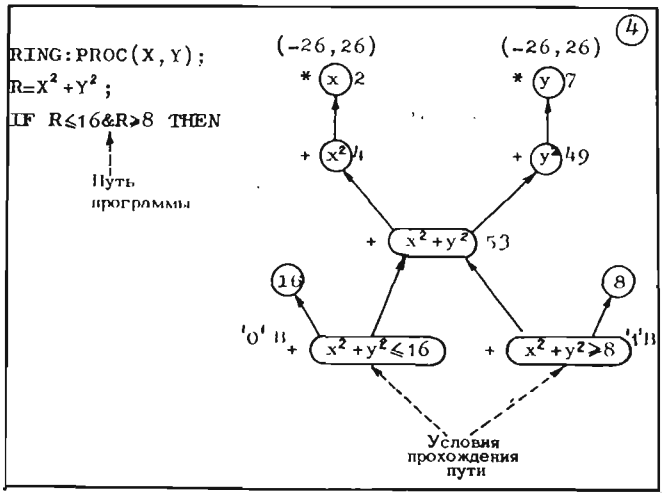
Шаг 7. В $ЛП_4$ последним является оператор конца программы. Это означает, что сгенерирован второй реализуемый стоп-путь $(ЛП_1, ЛП_3, ЛП_4) = (1, 3, 5, 6, 7)$.

Шаг 8. Поскольку не запоминались выходы со свойствами: «выход пока не реализован» и «выход отмечен как в принципе реализуемый» (таким выходом у нас был отмечен только выход (3,5) на шаге 2, и он использован на шаге 5), то генерация тестируемых путей закончена.

Если символьный интерпретатор и решатель неравенств в какой-нибудь момент не сумеют определить реализуемость, т. е. на запрос реализуемости по обоим выходам дадут отрицательный ответ, то отступаем по генерируемому пути назад




до первого выхода, который пока не реализован, но в принципе реализуем, и продолжаем генерацию по этому выходу. При отступлении возобновляем все измененные признаки.

Если на генерируемом пути не находится выход с вышеназванными свойствами, приступаем к выбору нового пути, как это было на шаге 5.

Если при выполнении критерия окончания генерации нет ни одного выхода, отмеченного как пока не реализованный, то мы получаем множество стоп-путей, которое покрывает все выходы.

**Символьное выполнение.** Интерпретатор предназначен для символьного выполнения пути программы в целях получения условий прохождения пути [15], т. е. системы неравенств над параметрами. В интерпретаторе использовано системное соглашение ПЛ/1, согласно которому логические отношения и логические операции выделяются среди традиционных операций (+, —, ...) лишь тем, что тип результата для них — BIT (1). На основании этого соглашения условие реализуемости задается очень просто: необходимо в совокупности всех символических значений отметить те, которые должны иметь значения '1'B, для того, чтобы данный путь был пройден.

Интерпретатор в основном работает с таблицей переменных и символическими значениями переменных; его основной работой является установка соотношений между переменными и их символическими значениями во время выполнения пути.

Покажем в общих чертах работу интерпретатора на примере пути (1, 3, 4, 5, 7) приведенной программы.

Перед началом выполнения пути устанавливаются начальные значения переменных: для параметров — их значения в начале выполнения, остальные переменные не инициализируются. Отношения между переменными и символическими значениями после начального присваивания показаны на рис. 2, *а*. Далее заданный путь интерпретируется по операторам.

Оператор 1: PROC (X, Y). Ничего не делается.

Оператор 3: IF X<17. Создается символическое значение $x<17$ типа BIT (1).

Оператор 4: THEN Y=1. Так как управление передано по THEN ветви, значение $x<17$ должно быть добавлено к условиям прохождения пути (пока множество условий пусто). Переменная Y получает символическое значение — константу 1. Ситуация после интерпретации оператора 4 показана на рис. 2, *б*.

Оператор 5: IF X>=17. Создается символическое значение $x>=17$ типа BIT (1).

Оператор 7: END TEST1. Так как управление передано по ELSE ветви предыдущего оператора IF, то строится отрицание значения $x>=17$ и это значение добавляется к условиям прохождения пути. Ситуация после выполнения последнего оператора пути показана на рис. 2, *в*.

После интерпретации пути условия его прохождения передаются решателю неравенства для определения реализуемости пути.

Конечно, показанный пример иллюстрирует работу символического интерпретатора очень поверхностно. При реализации интерпретатора возникают проблемы в интерпретации работы с массивами, с базированием, с вызовом подпрограмм. Все эти проблемы определенным образом решались, однако здесь на них останавливаться не будем.

**Решение неравенств.** Сначала выявляем существенные входные данные, т. е. те входные значения параметров, которые прямо или косвенно

использованы в условии прохождения пути. В примере существенным является только значение $x$ (на рис. 2, *г* оно отмечено звездочкой). Значение $y$ не существенно, поскольку от него не зависят передачи управления. Точно так же выявляем сложные существенные значения, т. е. такие, которые прямо или косвенно используются в условии прохождения пути, но которые получены в результате выполнения некоторых операций над другими значениями. На рис. 2, *г* сложные существенные значения отмечены знаком +. Знаком * и +, по существу, выделена интересующая нас система неравенств.

Поскольку необходимо найти числовые значения для существенных параметров (для $x$), то постараемся приблизительно оценить для них интервалы возможных значений, что, как правило, делается автором тестируемой программы. В примере, однако, предполагается, что пользователь таких интервалов не задавал. В таком случае система сама строит ограничения в виде $(-|a|-10, |a|+10)$, где $a$ — максимальная по модулю константа программы. В примере для $x$ строится интервал (—27, 27), как это показано на рис. 3, *а*.

Для получения численного решения система генерирует случайные значения из этих интервалов. В примере $x$ получает значение 26. Затем вычисляются значения для узлов, отмеченных знаком + (при $x<17$ получается '0'B, при $x\geqslant 17$—'1'B, а при $\neg(x\geqslant 17)$—'0'B). Так как не все узлы, отмеченные в условии прохождения пути, имеют значения '1'B, то решение не найдено. В таком случае генерируются новые случайные значения из интервалов и начинается проверка заново. Если за установленное число попыток решение угадать не удалось, то путь объявляется нереализуемым. На рис. 3, *б* показано, что решение $x=$ —24 найдено.

Рассмотренный нами пример тривиален — входное значение без арифметических преобразований использовано в условном операторе. На рис. 4 показан более сложный пример. В этом случае сначала угадываются значения для $x$ и $y$, потом вычисляются значения для $x^2$, $y^2$, $x^2+y^2$, $x^2+y^2\leqslant 16$, $x^2+y^2\geqslant 8$. Далее проверяется, имеет ли $x^2+y^2\leqslant 16$ и $x^2+y^2\geqslant 8$ значение '1'B. Если нет, то $x$ и $y$ присваиваются новые случайные значения и продолжается поиск.

Мы отказались от хорошо разработанных методов решения систем неравенств (например, методов линейного программирования) по следующим причинам:

— такие методы трудно приспособить для работы со строковыми данными и массивами. Метод проб и ошибок в этом отношении удобнее;

— ожидается, что системы, создаваемые в ходе анализа реальных программ, будут по своей логике очень простыми. В такой ситуации метод проб и ошибок может оказаться даже эффективнее.

Подобным образом поступали и другие авторы [14, 16]. Можно считать, что нами использовался упрощенный метод интервалов. Этот метод предполагает более разумную установку и корректировку интервалов возможных значений, что в нашей системе пока не реализовано.


1. *Барздинь Я. М., Бичевский Я. Я., Калниньш А. А.* Построение полных систем примеров для проверки корректности программ // Теория алгоритмов и программ.— Рига : Латв. гос. ун-т, 1974.— С. 152—187.
2. *Бичевский Я. Я.* Автоматическое построение систем примеров // Программирование.— 1977.— № 3.— С. 60—70.
3. *Калниньш А. А., Бичевский Я. Я., Барздинь Я. М.* Разрешимые и неразрешимые случаи проблемы построения полной системы примеров // Теория алгоритмов и программ.— Рига : Латв. гос. ун-т, 1974.— С. 188—205.
4. *Osterweil L. J., Fosdick L. D.* DAVE — a validation error detection and documentation system for FORTRAN programs // Software — Practice and Experience. — 1976. — 6.— P. 473—486.
5. *SMOTL* — a system to construct samples for data processing program debugging / J. Bicevskis, J. Borzovs, U. Straujums et al. // IEEE Trans. on Software Eng.— 1979.— 5, N 1.— P. 60—66.
6. *Muchnik S. S., Jones N. D.* (eds.). Program flow analysis: theory and applications.— Englewood Cliffs (N. J.): Prentice Hall, 1981.— 418 p.
7. *Касьянов В. Н., Поттосин И. В.* Технологические возможности оптимизации программ // Программирование.— 1980.— № 2.— С. 27—31.
8. *Culik K.* The cyclomatic number and the normal number of programs // ACM SIGPLAN Notices.— 1979.— **14**, N 4.— P. 12—17.
9. *Катков В. Л., Пилецкий И. И.* Подмножество языка ПЛ/1 для системного программирования // УСиМ.— 1983.— № 3.— С. 59—63.
10. *Марголин М. С.* О фильтрации параметров программных объектов // Программирование.— 1978.— № 4.— С. 33—37.
11. *Howden W. E.* Methodology for the generation of program test data // IEEE Trans. on Comput.— 1975.— **24**, N 5.— P. 554—560.
12. *Miller E. F., Melton R. A.* Automated generation of testcase datasets // Proc. Intern. Conf. on Rel. Software.— 1975.— P. 51—58.
13. *Boyer R. S., Elspas B., Levitt K. N.* SELECT — a formal system for testing and debugging programs by symbolic execution // Ibid.— P. 234—245.
14. *Ramamoorthy C. V., Ho S.-B. F., Chen W. T.* On the automated generation of test data // IEEE Trans. on Software Eng.— 1976.— 2, N 4.— P. 293—300.
15. *Борзов Ю. В.* Тестирование программ с использованием символического выполнения // Программирование.— 1980.— № 1.— С. 51—59.
16. *Derman E., Van Wyk C. J.* A simple equation solver and its application to financial modelling // Software — Practice and Experience.— 1984.— **14**, December.— P. 1169—1181.



Поступила 18.07.85
(после доработки — 03.02.86)

УДК 681.3.06

Ю. В. Борзов, Г. Б. Уртанс, В. А. Шимаров

# Выбор путей программы для построения тестов

## Введение

Обычная схема при тестировании программ включает следующие этапы: выбор критерия тестирования; выявление (построение) системы тестов, удовлетворяющей выбранному критерию; выполнение программы на полученной системе тестов; проверка результатов работы программы на тестах. В данной статье частично рассмотрим второй этап указанной схемы.

В [1, 2] из всех типов критериев тестирования наиболее распространенными признаются структурные (тестирования логики программы). Структурные критерии выбора тестов для отдельного модуля программы (*С*-критерии) [3] основаны на покрытии определенных структурных элементов программы (команд, ветвей, путей и т. д.) [4]. Наиболее распространенным является критерий *С*1 — покрытие всех ветвей программы, поэтому далее по умолчанию будем рассматривать именно этот критерий. Отношение числа покрытых элементов к числу всех элементов программы назовем степенью тестированности программы.

Систему тестов будем называть полной системой тестов (ПСТ) относительно заданного критерия тестирования, если достигается стопроцентная тестированность программы на этой системе тестов.

Покрывающим множеством путей (ПМП) назовем систему путей, содержащих в совокупности все элементы структуры программы согласно заданному критерию тестирования. При выполнении программы на ПСТ получаем ПМП. Таким образом, каждой ПСТ соответствует ПМП (обратное не всегда верно).

Проблема автоматического построения ПСТ рассматривалась в теоретических работах [5—7], имеются также экспериментальные разработки [8—11] систем автоматического построения ПСТ. Так как эта задача для нетривиальных классов программ алгоритмически неразрешима [6], в системах автоматического построения ПСТ используются различные эвристические алгоритмы.

Задача построения ПСТ состоит из получения двух взаимозависимых множеств: тестовых данных и путей (построение путей необходимо для определения полноты системы тестов). При построении необходимо взять за основу какой-то из этих объектов, т. е. нужно выбрать одну из двух возможностей:

— построить систему тестов и убедиться, что соответствующая система путей будет ПМП;

— построить ПМП и найти соответствующую ПСТ (если такая существует).

К методам первой группы можно отнести, например, все методы выбора тестов при тестировании программы как «черного ящика», существуют также методы специально для этих целей [12]. Однако значительно более развитыми на данный момент являются методы второй группы, основанные на построении ПМП. Построение ПСТ при этом состоит из двух частей — построения ПМП и определения для этого ПМП тестовых данных (если такие существуют). Данная статья посвящена методам, основанным на построении ПМП.

Задачи построения ПМП и нахождения тестовых данных можно пытаться решить последовательно. Методы построения ПМП при этом основываются только на структуре управления программы (управляющем графе программы — УГП). Очередной элемент ПМП строится без учета реализуемости рассматриваемых путей. Эту группу назовем *статическими методами* построения ПМП. Другая возможность построения ПСТ — одновременное построение ПМП и тестовых данных. При этом можно учитывать реализуемость или нереализуемость ранее рассмотренных путей или их частей. Эту группу будем называть *динамическими методами*. Третья группа заключается в выделении из множества всех путей множества всех реализуемых путей. После этого ПМП строится из полученного подмножества реализуемых путей. Эту группу назовем *методами реализуемых путей*.

Напомним несколько определений. Путь программы — последовательность операторов, по которой формально возможна передача управления в программе. Путь реализуемый, если существуют входные данные, на которых программа выполняет именно этот путь. Ограничения на входные данные, достаточные для того, чтобы выполнялся заданный путь,— это условия реализуемости пути (их можно, в частности, получить при символическом выполнении пути как систему неравенств над входными значениями программы [13]). Начальный путь — это путь, начинающийся в точке входа в программу. СТОП-путь — это путь, заканчивающийся выходом из программы.

## Статические методы построения систем путей

**Основные положения.** Поскольку статические методы построения ПМП, как правило, основаны на структуре передачи управления, то и основные понятия несколько видоизменяются с учетом теоретико-графовой терминологии. Так, пути будут выражаться главным образом в терминах вершин и дуг УГП. Вершинам УГП соответствуют линейные последовательности операторов программы (в принципе можно рассматривать и от-

дельные операторы), а дугам — передачи управления между ними. Кроме того, иногда вместо СТОП-путей будем говорить о путях тестирования, что часто будет означать то же самое.

Многие статические методы построения ПМП в той или иной степени имеют аналоги среди динамических методов (реже — среди методов реализуемых путей), однако подход к построению без учета условий реализуемости иногда позволяет решать интересные как в теоретическом, так и в практическом плане задачи.

**Оптимизационные методы выбора путей.** В особенности это относится к оптимизационным задачам и методам построения путей. Наиболее строгой и конкретной задачей является задача построения минимального по числу путей ПМП.

В случае критериев $C0$ и $C1$ эта задача впервые достаточно полно исследована в [14, 15]. Предложенные там методы применимы к графам без циклов, поэтому УГП $G$ должен быть преобразован в соответствующий ему граф Герца $G1$ [16], в котором вершинами являются сильносвязные компоненты (ССК) графа $G$, т. е. подграфы, включающие взаимно достижимые вершины УГП. Методы построения ПМП требуют $O(n^{5/2})$ и $O(n^2)$ операций, базируясь соответственно на алгоритмах нахождения максимального паросочетания и минимального потока (считается, что $e=O(n)$). Здесь и далее $n$ и $e$ — число вершин и дуг УГП.

Позднее для ациклических графов был предложен более эффективный алгоритм построения ПМП [17], который, правда, нуждается в незначительных уточнениях. Тем не менее он требует всего $O(i_e(G)\nparallel e)$ операций, где $i_e(G)$ — максимальное число попарно недостижимых дуг графа $G$ [14]. Быстродействие алгоритма обеспечивается тем, что при построении пути можно исключать из графа те вошедшие в путь дуги, которые не мешают покрытию других дуг, не обрабатываемых ранее.

При использовании последнего алгоритма на первый план выдвигаются проблемы развертки ССК, т. е. детализации той части пути, которая касается прохождения по ССК. Считается, что ССК покрывается одним путем, поэтому длина развертки ССК может достигать $O(e^2)$ [18], т. е. даже простой просмотр пути в $G$ может потребовать больше операций, чем построение ПМП для $G1$. Так как анализ путей для определения тестовых данных проводится вручную, для большей свободы выбора иногда полезнее представлять пути без развертки ССК.

Чтобы избежать проблемы развертки ССК, в [15] предложена другая операция приведения УГП к ациклическому виду — замена цикла условным оператором. Однако тогда может существовать ПМП, содержащее меньшее число путей, чем полученное при применении метода, а поэтому говорить о минимальности ПМП уже нельзя.

Интересным представляется подход [19], при котором сначала строится ПМП для ациклической части УГП (без его явного преобразования к ациклическому виду), а затем пути «дополняются» циклами. В другом аналогичном подходе [20] простые пути комбинируются с путями минимальной суммарной длины, покрывающими все циклы.

**Эвристические (статические) методы выбора путей.** С практической точки зрения часто могут оказаться полезными эвристические методы построения ПМП. Один из таких методов [21] на самом деле предназначался для иной области применения (конечные автоматы) и заключается в удлинении начального пути за счет непокрытых дуг до тех пор, пока это возможно. Некоторые появившиеся позднее алгоритмы [22, 23], по сути, являются видоизменением этого метода. Так, основные изменения, сделанные в [22], направлены на устранение возможности получения чрезмерно длинных путей.

И статические, и динамические эвристические методы (рассматриваемые ниже) имеют много общего. Отличие же в том, что статические по своей природе не допускают возвратов после «неудачного» продолжения начального пути, а потому имеют достаточно низкую временную сложность.

Самые быстрые (но непрактичные) методы на основе поиска в ширину или в глубину могут иметь линейную сложность, тогда как эффективная реализация рассмотренных выше методов [22] требует $O(e^2)$ операций. Сам по себе метод из [22] не лишен серьезных недостатков, так как, например, игнорирует число повторений циклов в программе. Однако он достаточно успешно применяется как отдельный этап в более общем алгоритме, относящемся к следующей группе методов [24].

**Факторизованные методы.** Они основаны на представлении структуры управления в виде иерархически вложенных подграфов специального вида (как правило, конструкций структурного программирования).

Обычно факторизация применяется для уменьшения сложности алгоритмов, но в нашем случае это объясняется другими причинами: во-первых, возможностью более удачного представления путей (например, скобочное представление циклов); во-вторых, необходимостью проводить тестирование программы от общих свойств к частным, сначала проверяя внешние логически связанные части, затем последовательно — ее внутренние составные элементы [19]; в-третьих, возможностью более или менее эффективного учета простейших видов условий реализуемости, введенных в [14].

Одним из первых факторизованных представлений является интервальное представление УГП [4], применяемое при глобальной оптимизации программ. Главным свойством интервала с един-

ственным входом $h$ является то, что все его циклы содержат $h$. На этом и основан метод из [25], который на уровне каждого отдельного интервала можно отнести к эвристическим методам. Его преимуществами являются эффективность и удобный способ учета циклов.

Однако гораздо более наглядным и не менее эффективным является использование такого свойства многих программ, как структурированность, что значительно облегчает построение ПМП [15].

Поскольку не все программы полностью составлены только из допустимых конструкций, имеет смысл применять некоторые компромиссные факторизованные представления потока управления «почти» структурированных программ.

Таким представлением является структурное дерево программы (СДП) [22], вершинам которого соответствуют структурированные и неструктурированные конструкции, вершины УГП. Дугам же соответствуют отношения вложенности между конструкциями. Каждой вершине СДП (точнее, соответствующей ей конструкции) приписывается набор типовых выполнений, причем для последовательности и конструкции выбора они совпадают со всеми путями выполнения. Для циклов и неструктурированных конструкций определение множества типовых выполнений может быть неоднозначным и зависеть не только от конкретного критерия тестирования, но и от степени учета условий реализуемости путей.

Обозначим $\mathrm{path}(x)$ — путь, соответствующий некоторому типовому выполнению $x$; $\mathrm{TB}(i)$ — множество всех возможных типовых выполнений вершины (конструкции) $i$. Для учета динамики покрытия элементов $\mathrm{TB}(i)$ введем два его подмножества: $\mathrm{TB1}(i)$ состоит из «непокрытых» элементов $\mathrm{TB}(i)$, а $\mathrm{TB2}(i)$ содержит такие типовые выполнения $x$, что для некоторой вершины $j$, входящей в $\mathrm{path}(x)$, либо $\mathrm{TB1}(j)$, либо $\mathrm{TB2}(j)$ непусты. Такие вершины $j$ назовем активными и обозначим это свойство предикатом $\mathrm{act}(j)$.

Построение ПМП для критерия $C1$ фактически сводится к покрытию для каждой вершины $i$ всех типовых выполнений из множества $\mathrm{TB1}(i)$, причем в начале тестирования $\mathrm{TB1}(i)=\mathrm{TB}(i)$. Алгоритм нахождения одного пути в упрощенном виде заключается в следующем:

1) занести в текущий путь $p$ корень СДП $r$;

2) детализировать вершины пути $p$ до тех пор, пока в нем останутся только вершины УГП (для них тип равен simple). Заметим, что, поскольку вершина $r$ представляет всю программу в целом, путь $p$ с самого первого шага построения является начальным СТОП-путем. В этом существенное отличие данного метода от рассмотренных выше, что позволяет решать задачу построения ПМП с некоторыми ограничениями на вхождение пар вершин или дуг в путь (например, «обязательные пары» из [14, 15]).

Опишем алгоритм формально.

```
Покрытие: proc;
Инициализация;
de while (act(r));
  p=r;
  poz=1;
  do while (poz<=length(p));
    /*детализация в позиции poz*/
    i=p(poz);
    if type(i)=simple
    then do; poz=poz+1; корректировка; end;
    else if act(i)
      then do;  выбрать 1(x);
                заменить (poz, path(x));
        end;
      else do;  выбрать 2(x);
                заменить (poz, path (x));
               poz=poz+length(path);
        end;
    end; /*очередной путь построен*/
  end; /*построено ПМП*/
end покрытие;
```

В процедуре ИНИЦИАЛИЗАЦИЯ устанавливаются множества $\mathrm{TB}(i)$, $\mathrm{TB1}(i)$, $\mathrm{TB2}(i)$ и признак активности $\mathrm{act}(i)$ для всех вершин $i$, начиная с листьев СДП и кончая корнем $r$.

В процедуре ВЫБРАТЬ1 выбирается элемент (типовое выполнение), входящий либо во множество $\mathrm{TB1}(i)$, либо во множество $\mathrm{TB2}(i)$, причем в первом случае он сразу удаляется из $\mathrm{TB1}(i)$. Следует отметить, что если типовое выполнение $x$ «покрывает» с точки зрения выбранного критерия тестирования типовое выполнение $y$, то надо $y$ удалить вместе с $x$.

Если при выборе $x$ приоритет отдается множеству $\mathrm{TB1}(i)$, то получаем подобие «нисходящего» тестирования программы. Если же в первую очередь выбирать $x$ из $\mathrm{TB2}(i)$, то имеем «восходящее» тестирование, при котором сначала тестируются самые вложенные конструкции, а затем уже — их охватывающие.

Процедура ВЫБРАТЬ2 может иметь две модификации. Дело в том, что для неактивной вершины СДП иногда неважно, как происходит выполнение соответствующего участка программы с одним входом и одним выходом. В этом случае $\mathrm{path}(i)=\{k, \ldots, l\}$, где $k$ и $l$ — вход и выход соответствующей конструкции, а средняя «вершина» означает произвольный путь от $k$ до $l$ в этой конструкции.

Если же требуется полный путь в УГП, то выбирается произвольный элемент множества $\mathrm{TB}(i)$.

В процедуре КОРРЕКТИРОВКА при достижении листа СДП производится итеративное уточнение множества $\mathrm{TB2}(i)$ и признака $\mathrm{act}(i)$ для вершин — предшественников в СДП.

Процедура ЗАМЕНИТЬ используется для вставки детализации вершины $i$ вместо самой вершины, расположенной в позиции poz.

Трудоемкость алгоритма построения путей, если не учитывается детализация неструктурированных конструкций и набор типовых выполнений простейший, равна $O(g \times l)$ [24], где $g$ — число путей, а $l$ — максимальная длина пути. Обработка неструктурированных конструкций требует $O(e^2)$ операций. Для них понятие типового выполнения не фиксируется, просто по мере необходимости строятся пути, покрывающие дуги конструкции, либо активные вершины по алгоритму из [22]. Если не детализировать неактивные вершины, то длина пути $l=O(n)$ и общая трудоемкость $O(e^2)$.

**Ручная интерпретация путей.** Итак, рассматриваемый подход позволяет получать целые семейства (в терминологии [4] — классы) эквивалентных с точки зрения критериев $C1$ путей, что в значительной степени облегчает окончательный их выбор. К уже упомянутой ранее скобочной записи циклов, позволяющей самому программисту задавать необходимое число итераций, добавляется возможность не детализировать подпути, проходящие по уже тестированным фрагментам программ с одним входом и одним выходом (т. е. не детализировать «оттестированные» вершины СДП). Действительно, если в указанных фрагментах не перевычисляются операнды условных выражений, расположенных вне фрагментов, то их отсутствие не влияет на выбор тестовых данных.

С точки зрения критерия $C1$ заключительную часть пути, не содержащую впервые именно этим путем покрываемых дуг УГП, также не обязательно анализировать. Больше того, в случае возникновения трудностей при построении тестов можно оставлять начальную часть пути, содержащую хотя бы одну впервые покрытую дугу. Последняя возможность в пределе дает стратегию тестирования префиксных путей [10, 26].

Суть ее заключается в следующем: если известен тест, покрывающий начальный (префиксный) путь $p=(n_1, \ldots, n_j, k_1)$, а дуга $(n_j, k_2)$ не покрыта, то иногда легко найти такую незначительную корректировку теста, которая может вызвать прохождение префиксного пути $p'=(n_1,\ldots,n_j, k_2)$ ($p'$ можно назвать обращением $p$ [26]). Таким образом, за вполне допустимое число тестовых прогонов (выполняемых явно или, реже, моделируемых) можно в ряде случаев достигнуть полного покрытия графа, причем с целью оптимизации затрат на тестирование в [26] предложено выбирать кратчайшие префиксные пути.

Завершая описание статических методов выбора путей, следует отметить, что многие из них взаимно дополняют друг друга и комплексное их использование способствует более эффективной организации структурного тестирования. Поэтому целесообразно в системах генерации тестов иметь несколько возможностей, как, например, сделано в системе «Автограф» [24], включающей многие из описанных возможностей на базе всех видов статических методов (минимальное ПМП без развертки ССК, эвристический алгоритм [22] и описанный факторизованный метод построения на базе СДП).

## Динамические методы построения ПМП

**Основные положения.** Динамические методы построения ПМП (далее стратегии) в отличие от статических учитывают не только структуру передачи управления в программе, но и реализуемость выбранных путей. Результатом работы динамических методов является не просто ПМП, а покрывающее множество реализуемых путей.

Основной идеей динамических методов является подсоединение к начальным реализуемым отрезкам путей дальнейших их частей (до построения начальных СТОП-путей) таким образом, чтобы, во-первых, не терять при этом реализуемости вновь полученных путей; во-вторых, покрыть требуемые элементы структуры программы.

Для проверки реализуемости пути можно использовать символический интерпретатор пути программы с решателем условий реализуемости, однако для этого также может быть использовано любое другое средство, определяющее реализуемость пути (включая диалог с пользователем). На проблемах определения реализуемости здесь останавливаться не будем, предполагая, что у нас есть «оракул», выдающий достоверную информацию о реализуемости.

Для обеспечения покрытия требуемых элементов используется алгоритм (стратегия) выбора продолжения начальных путей, который работает в зависимости от уже покрытых и еще непокрытых элементов, структуры и семантики программы. Стратегии принципиально различаются между собой именно способом выбора продолжений начальных путей. Конечно, можно остановиться на простом переборе всех возможных путей, не заботясь о порядке перебора. Существуют классы программ, для которых такие алгоритмы хорошо работают. Это, например, программы без сложных циклов, с небольшим (часто конечным) числом относительно коротких путей. Однако при возрастании сложности логики программ (особенно с вложенными циклами, с неограниченным числом итераций, взаимосвязанными условиями числа итераций циклов) и даже объема программ простой перебор часто не дает удовлетворительных результатов.

Отметим также, что определение реализуемости пути требует достаточно больших ресурсов, и простой перебор может привести к их перерасходу. Для этих случаев необходимо использовать эвристические методы выбора последовательности рассматриваемых путей.

**Эвристические методы выбора путей.** Эвристику стратегии определяет очередность выбора продолжений начальных реализуемых путей (НРП), целесообразность продолжения построения ПМП после построения очередного элемента ПМП. Остановимся на этих проблемах подробнее.

Очередность выбора продолжений НРП определяет, пути какого вида и в какой последовательности будут строиться. Дадим несколько утверждений, которые можно выбрать для разработки эвристики:

1) строить как можно более короткие пути, т. е. двигаться как можно быстрее к оператору выхода из программы;

2) строить как можно более длинные пути;

3) выходить из цикла при первой возможности;

4) пытаться перед выходом из цикла покрыть все его элементы;

5) пытаться сделать, как минимум, $n$ итераций в цикле перед выходом из него;

6) идти сначала по «семантически более естественным» ветвям (например, по ветви ввода данных при операторах ввода) и только после этого по «менее естественным» (по ветви END-FILE);

7) идти по ранее непокрытым ветвям;

8) строить одновременно только один путь (от начала до конца), затем переходить к построению следующего;

9) строить одновременно несколько путей, продолжая наиболее перспективный на данный момент.

Хотя некоторые из утверждений попарно противоречивы, для каждого из них можно указать достаточно разумные обоснования. Эвристика может содержать сочетание указанных или других идей. Возможно также изменение принципов работы стратегии в зависимости от структуры программы, сложности условий реализуемости, временных ресурсов для построения и т. д.

Стратегия завершается успешно, если получено ПМП или исчерпаны все продолжения НРП, т. е. просмотрены все реализуемые пути. В первом случае получено ПМП, во втором улучшить степень тестированности программы по сравнению с полученной не удастся, так как рассмотрены все реализуемые пути программы.

Необходимость определить целесообразность продолжения построения объясняется тем, что не всегда удается автоматически получить стопроцентное покрытие. Это может произойти по следующим двум причинам: некоторые структурные элементы в принципе нереализуемы, т. е. не существует входных данных, при которых покрывается часть структурных элементов; для покрытия некоторых элементов необходимо построить слишком длинный и сложный путь.

В первом случае покрыть элементы невозможно, во втором можно продолжить построение для полного покрытия. Сложность проблемы состоит в том, что во многих случаях невозможно определить, по какой из причин элемент еще не покрыт. При попытке решения второй проблемы может к тому же потребоваться очень большой перебор вариантов, причем оценить число вариантов перед началом работы стратегии трудно (если вообще возможно).

Решение о прекращении работы может основываться на утверждениях следующего типа: исчерпаны временные ресурсы работы; рассмотрены все реализуемые пути длиною $x$; все циклы пройдены хотя бы $x$ раз; все реализованные ветви пройдены хотя бы $x$ раз; получено покрытие элементов на $x$ процентов; за последние $x$ итераций цикла стратегии не получено ни одного СТОП-пути, покрывающего ранее непокрытые элементы.

**Общая схема работы стратегий.** При описании стратегий (независимо от используемых в них эвристик) будем применять следующие рабочие множества объектов: ПМП — покрывающие множество путей; НЭ — множество непокрытых элементов; ЭПНП — множество элементов, покрытых начальными путями; ЭПСП — множество элементов, покрытых построенными СТОП-путями; НРП — множество начальных реализуемых путей — кандидатов для дальнейшего построения путей; ВПЭ — множество всех покрываемых элементов.

Основными объектами являются элементы НРП, содержащие следующую информацию о начальных путях: начальный путь — НРП.НП; состояние программы после символьного выполнения данного пути (значения переменных, условия реализуемости) — НРП.СП; покрытые элементы — НРП.ПЭ; приоритет пути используется для определения очередности выбора продолжений НРП, способ вычисления приоритета задает используемая эвристика — НРП.ПП.

Для работы с этими множествами введем несколько обозначений и операций. Будем использовать стандартные операции над множествами: объединение «+», сечение «$\times$» и разность «/».

Предположим, что все элементы множеств перенумерованы. Тогда НРП($i$) — $i$-й элемент множества НРП. Число элементов НРП обозначим !НРП!. Первым элементом множества НРП всегда будет начальный путь от входа в программу до первого реализуемого ветвления — оператора с более чем одним реализуемым выходом.

Введем предикат

$$\text{stop}\ (\text{НРП}(i)) = \begin{cases} \text{true}, & \text{если НРП}(i)\ \text{СТОП-путь;} \\ \text{false} & \text{в других случаях.} \end{cases}$$

Введем функцию succ (НРП($i$), $j$) = НРП($k$), где НРП($k$) — продолжение НРП($i$) по $j$-му реализуемому выходу до следующего реализуемого ветвления или выхода из программы; допустимое значение для определенного элемента НРП (т. е. число реализуемых продолжений начального пути) обозначим $re$ (НРП($i$)).

Стратегия представляет собой цикл, при каждом проходе которого строится продолжение определенного элемента из НРП. Если в качестве продолжения получается СТОП-путь, покрывающий ранее непокрытые элементы, то он добавляется к ПМП. После очередного прохода цикла определяется, необходимо ли продолжать построение.

Перед первым выполнением цикла рабочие объекты стратегии заполняются следующим образом:

1) ПМП — пустое;
2) НЭ — все структурные элементы;
3) ЭПНП — пустое;
4) ЭПСП — пустое;
5) НРП — содержит единственный элемент: путь от входа программы до первого оператора с несколькими реализуемыми выходами.

Действия, выполняемые при проходе цикла стратегии.

1. Выбираем из НРП путь с наибольшим приоритетом.

2. Если это СТОП-путь, добавляем его к ПМП (если данный элемент покрывает хотя бы один ранее непокрытый элемент НЭ).

Если это не СТОП-путь, в НРП добавляем все его реализуемые продолжения до следующего реализуемого ветвления или выхода из программы. Выбранный ранее путь из НРП удаляем.

3. Определяем целесообразность продолжения построения ПМП.

Все описанные действия сопровождаются соответствующей корректировкой рабочих множеств.

**Более формальное описание стратегии.**

```
Стратегия: proc;
/* начальное присваивание */
ПМП=0,  НЭ=ВПЭ;  ЭПНП=0;  ЭПСП=0;  НРП=
=(НРП(1));
/* цикл */
do until (НЭ=0 ! НРП=0 ! «нецелесообразно продол-
жать»);
  e=1;
  /* нахождение элемента с наивысшим приоритетом */
  do i=1 to !НРП!;
    if НРП.ПП(i)>НРП(e) then e=i;
    end;
  if stop (НРП(e))
  then do; /* СТОП-путь */
    if НРП.ПЭ(e) * НЭ^=0
    then do /* покрыты новые элементы */
      ПМП=ПМП+{НРП(e).НП} ; /* корректировка */
      НЭ=НЭ/НРП(e).ПЭ ; /* множеств */
      ЭПНП=ПЭ / НРП(e).ПЭ;
      ЭПСП=ЭПСП+НРП(e).ПЭ;
      end;
    НРП=НРП / {НРП(e)} ;
  else do; /* не СТОП-путь */
    /* добавление продолжений */
    do i=1 tore (НРП(e));
      НРП=НРП+{succ(НРП(e),i)};
      ЭПНП=ЭПНП+succ (НРП(e), i). ПЭ;
      end;
    /* удаление выбранного элемента */
    НРП=НРП / {НРП(e)} ;
    end;
  end
end стратегия ;
```

**Примеры стратегий.** Рассмотрим несколько примеров эвристик, используемых в стратегиях.

*Стратегия первой версии системы тестирования ТЕСТГЕН.* Система ТЕСТГЕН [27] предназначена для автоматического построения ПСТ программ на подмножестве ПЛ-1. В качестве критерия тестирования выбран критерий *С1*. Для выбора продолжения НРП используются утверждения 2, 6—8 из раздела «Эвристические методы выбора путей».

Пути строятся последовательно: один путь строится до конца (получения СТОП-пути), затем берется первый (в порядке построения) НПР, последняя ветвь которого еще не покрыта ни одним СТОП-путем, и строится новый путь. Алгоритм заканчивается, если не существует элементов НРП, содержащих еще непокрытые ветви программы. Так как одновременно строится только один путь, в качестве следующего рассматриваемого элемента берется какое-либо из его продолжений. Далее выбирается еще нереализованное продолжение. Если таких несколько, сначала выбирается ветвь THEN или ветвь ввода данных; если упомянутых нереализуемых продолжений нет, то ветви выбираются поочередно.

*Стратегия второй версии системы тестирования ТЕСТГЕН.* Система предназначена для автоматического построения ПСТ программ, написанных на языке ПЛ-1. В качестве критерия тестирования выбран критерий *С1*. Для выбора продолжений НРП используются утверждения 1, 7, 9.

Приоритет элемента НРП вычисляется по формуле

$$К1 * ЧПЭ - К2 * РНЭ - К3 * ДНП,$$

где К1, К2, К3 — настраиваемые параметры: ЧПЭ — число элементов, покрытых данным путем и не входящих в ЭПСП; РНЭ — расстояние до ближайшего непокрытого элемента (в качестве расстояния используется минимальное число операторов ветвления от конца элемента НРП до непокрытого элемента); ДНП — длина пути, т. е. число операторов ветвления на данном пути.

Параметры К1, К2, К3 выбираются таким образом, чтобы наибольший вес имело слагаемое К2 * РНЭ. Одновременно строится несколько путей, при этом в каждый момент продолжается тот, который ближе всего к ранее непокрытому элементу. Остальные два слагаемых должны предотвратить «зацикливание» алгоритма, когда один из элементов НРП находится очень близко к непокрытому элементу, но путь к этому элементу в принципе нереализуем.

Алгоритм завершается, если за К4 последних итераций цикла стратегии: не получен ни один новый элемент ПМП; максимальный приоритет элементов НРП не увеличился; К4 — настраиваемый параметр.

## Методы реализуемых путей

**Основные идеи.** Методы реализуемых путей характеризуются последовательным выявлением реализуемых путей программы и построением

ПМП из них. Для этого строится граф реализуемости программы (ГРП), содержащий только все реализуемые пути программы. В ГРП все выходы всех условных ветвлений при любом пути достижения этих ветвлений реализуемы. После получения ГРП его покрытие можно осуществить любыми ранее описанными статическими методами. При этом все полученные пути будут реализуемы (по определению ГРП). ГРП содержит в себе все структурные элементы программы, покрытие которых возможно реализуемыми путями.

Построение ГРП основано на понятии состояния программы. Состояние программы после выполнения пути программы содержит значения переменных (как правило, значения, получаемые при символическом выполнении программы) и условия реализуемости в конечной точке пути. Состояние программы после выполнения пути $x$ обозначим $S(x)$. Состояние может содержать не всю упомянутую информацию. Информация, содержащаяся в состоянии, должна быть достаточной для того, чтобы всегда выполнялось следующее утверждение.

Пусть заданы пути $x$, $y$, заканчивающиеся одним и тем же оператором, и их продолжение $z$. Тогда из $S(x)=S(y)$ следует, что $x+z$ и $y+z$ реализуемы или нереализуемы одновременно (обозначим $x+z$ — сцепление путей $x$ и $z$) и $S(x+z)=S(y+z)$.

Идея построения ГРП состоит в описании всех переходов от одних состояний программы в другие и основана на предположении о конечности числа состояний программы. Конечность числа состояний программы доказана в [5] для программ базового языка, являющегося простой моделью языков обработки последовательных файлов.

Можно надеяться, что число состояний конечно и для программ, не содержащих сложных преобразований тех данных, которые появляются в условных операторах.

Схему построения ГРП можно описать следующим образом.

Выбираем систему элементарных путей (ЭП) программы — конечных отрезков путей, из которых сцеплением можно получить любой путь программы. В качестве ЭП можно использовать отдельные команды, линейные пути (ветви), пары ветвей и т. д.

Для всех ЭП и всех состояний программы, которые могут быть получены в начале выполнения ЭП (т. е. для всех допустимых пар — ЭП, состояние), строим состояние программы после выполнения ЭП. Вновь полученное состояние становится допустимым для ЭП, находящихся непосредственно за выполненным ЭП.

После построения всех переходов отобразим их в графе, вершинами которого являются упомянутые пары (ЭП, состояние), а дугами — переходы между ними. Полученный граф — ГРП, так как он содержит только все реализуемые пути в программе (это следует из определения состояния программы).

**Примеры методов реализуемых путей.** *Автоматическое построение ПСТ для программ на базовом языке.* В [4, 6] доказана алгоритмическая разрешимость проблемы построения ПСТ для базового языка и дан алгоритм построения. Как уже упоминалось, алгоритм основан на конечности числа состояний программы, которая для программ этого языка определяется ограниченным количеством доступных в программе классов существенно различных значений переменных (нет арифметики).

В алгоритме в качестве ЭП (см. схему реализуемых методов) используются команды, а в качестве ГРП строится дерево реализуемости — ациклический подграф ГРП, который удобно использовать для построения путей.

Опишем схему работы алгоритма.

1. Строим развертку программы в виде дерева. Для этого берем первую команду программы, строим начальное состояние программы и заносим эту пару в нулевой ярус дерева.

2. После построения $k$-го яруса дерева строим $(k+1)$-й ярус: для всех пар (команда, состояние) $k$-го яруса в $(k+1)$-й ярус заносятся все реализуемые продолжения (команда, состояние) из этих команд при этих состояниях. Элементы $k$-го яруса соединяются с их продолжениями.

Продолжение (пара: команда, состояние) не заносится в $(k+1)$-й ярус, если на пути от нулевого до $(k+1)$-го яруса добавляемого элемента уже находится такая же пара, как на $(k+1)$-м уровне. В этом случае дугу от $k$-го в $(k+1)$-й ярус будем называть оборванной по повторению (для построения ГРП оборванную дугу следует направить на повторяющуюся вершину).

*Система СМОТЛ.* Идеи построения ПСТ для базового языка использованы в экспериментальной системе автоматического построения тестов СМОТЛ [9]. Эта система предназначена для тестирования по критерию $C1$ программ, написанных на СМОД (Система Макрокоманд Обработки Данных) — КОБОЛ-подобном языке для ЭВМ «Минск-32». Система завершена в 1976 г.

В качестве множества ЭП в системе используются все пути между существенными операторами (СО) программы. Множество операторов назовем множеством СО, если на любом пути программы между любыми повторяющимися операторами находится хотя бы один оператор из этого множества. Использование хорошо выбранного множества СО позволяет минимизировать число состояний программы и тем самым повысить эффективность тестирования.

Хотя СМОД является языком, для которого в общем случае задача построения ПСТ алгоритмически неразрешима, авторы системы надеялись на то, что в большинстве случаев число состояний будет конечно. Если при построении число состояний оказывается больше, чем позволяют обработать доступные ресурсы памяти, построение новых состояний прекращается и покрывается частично полученный ГРП. В последнем случае алгоритм СМОТЛ можно считать улучшенным вариантом простого перебора НРП в динамических методах построения ПМП. Оптимизация перебора при этом заключается в использовании состояния программы, позволяющего прекратить продолжение пути при повторении состояния на нем.

## Сравнительный анализ методов

Каждый из описанных методов имеет свои достоинства и недостатки. Считаем, что методы не следует противопоставлять, а применять их совместно, используя достоинства и компенсируя недостатки каждого из них.

Достоинство статических методов состоит в сравнительно небольших требуемых ресурсах как при использовании, так и при разработке (последнее также важно потому, что пока промышленных систем построения ПМП для основных языков программирования нет). Однако продукция этих систем (ПМП) может содержать заранее непредсказуемый процент брака (нереализуемых путей), который к тому же трудно исправить, так как пока не существует алгоритмов дополнения систем путей в случае нереализуемости части путей. Кроме того, в этих системах переход от ПМП к ПСТ пользователь должен осуществить вручную, а эта работа достаточно трудоемка и нередко малоприятна.

Динамические методы требуют значительно больших ресурсов как при разработке, так и при эксплуатации, однако увеличение этих затрат происходит в основном за счет разработки и эксплуатации аппарата определения реализуемости пути (символический интерпретатор, решатель неравенств). Достоинство этих методов в том, что их продукция имеет некоторый качественный уровень — реализуемость путей. Помимо путей тестирования, системы динамических методов выдают также тестовые значения (результат работы аппарата реализуемости), что значительно уменьшает «ручной» труд программиста, указанный при анализе статических методов.

Методы реализуемых путей дают наиболее качественный результат, однако применение этих методов для широких классов программ проблематично, и в связи с большими затратами при разработке их промышленная реализация в ближайшем будущем маловероятна.

Возможно, что наиболее приемлемой окажется такая последовательность действий при построении ПСТ.

1. Написание системы случайных и/или существенных с точки зрения программиста (или заказчика) тестов и определение соответствующей системы покрывающих путей.

2. Дополнение полученной системы путей статическими методами и определение реализуемости полученного дополнения (с возможным итерированием внутри данного пункта).

3. Покрытие оставшейся части программы с помощью динамических методов.

Заметим, что построение ПСТ может завершиться на любом шаге указанной схемы.


1. *Майерс Г.* Искусство тестирования программ.— М.: Финансы и статистика, 1982.— 176 с.
2. *Бичевский Я. Я., Борзов Ю. В.* Тестирование программ ЭВМ.— Рига : ВЦ ЛатвГУ, 1986.— 101 с.
3. *Miller E. F.* Coverage measure definitions reviewed // Testing Techn. Newsl.— 1980.— 3, N 4.— P. 6.
4. *Paige M. R.* On partitioning program graphs // IEEE Trans. Softw. Eng.— 1977.— 3, N 6.— P. 386—393.
5. *Барздинь Я. М., Бичевский Я. Я., Калниньш А. А.* Построение полных систем примеров для проверки корректности программ // Теория алгоритмов и программ.— Рига : ЛатвГУ, 1974.— С. 152—187.
6. *Калниньш А. А., Бичевский Я. Я., Барздинь Я. М.* Разрешимые и неразрешимые случаи проблемы построения полной системы примеров // Там же.— С. 188—205.
7. *Бичевский Я. Я.* Автоматическое построение систем примеров // Программирование.— 1977.— № 3.— С. 60—70.
8. *Clarke L. A.* A system to generate test data and symbolically execute programs // IEEE Trans. Softw. Eng.— 1976.— 2, N 3.— P. 215—222.
9. *SMOTL* — a system to construct samples for data processing program debugging / J. Bicevskis, J. Borzovs, U. Straujums et al.// Ibid.— 1979.— 5, N 1.— P. 60—66.
10. *Правильщиков П. А.* Построение тестов для программ // Автоматика и телемеханика.— 1977.— № 3.— С. 147—160.
11. *Howden W. E.* Methodology for the generation of program test data // IEEE Trans. on Comput.— 1975.— 24, N 5.— P 554—560.
12. *Kundu S.* SETAR — a new approach to test case generation // Infotech St. of the Art Rep. Softw. Testing.— 1979.— N 2.— P. 161—186.
13. *Уртанс Г. Б.* Функции путей программ // Программирование.— 1987.— №4.— С. 69—78.
14. *Ntafos S. C., Hakimi S. L.* On parh cover problems in digraphs and its applications to program testing // IEEE Trans. Softw. Eng.— 1979.— 5, N 5.— P. 520—529.
15. *Ntafos S. C., Hakimi S. L.* On structured rigraohs and program testing // IEEE Trans. on Comput.— 1981.— 30, N 1.— P. 67—77.
16. *Евстигнеев В. А.* Применение теории графов в программировании.— М.: Наука, 1985.— 352 с.
17. *Иыуду К. А., Арипов М. М.* Тестирование программы на основе минимального покрытия его графа // УСиМ.— 1985.— № 4.— С. 69—71.
18. *Gabow H. N., Maheshwary S. N., Osterweil L. G.* On two problems in the generation of program test paths // IEEE Trans. Softw. Eng.— 1976.— 2, N 3.— P. 227—231.
19. *Позин Б. А.* Метод структурного построения тестов для отладки управляющих программ // Программирование.— 1980.— № 2.— С. 62—69.
20. *Masuyama H., Ichimori T.* Optimum set of paths which pass all archs of a graph // Trans. IECE Japan.— 1986.— 69, N 1.— P. 6—8.
21. *Грунский И. С., Сперанский Д. В.* О покрытии графа множеством путей, исходящих из одной вершины // Кибернетика.— 1974.— № 1.— С. 29—34.
22. *Катков В. Л., Шимаров В. А.* Статический анализ программы с помощью ее управляющего графа // УСиМ.— 1986.— № 2.— С. 89—92.
23. *Торопов Д. И.* О построении тестирующих маршрутов программы // Там же.— 1985.— № 4.— С. 69—71.
24. *Шимаров В. А., Головня Л. А., Дулькин Л. Б.* Анализ управляющего графа СИПЛ-программ // Программное обеспечение ЭВМ. Инструментальный комплекс РИТМ для разработки программного обеспечения ЕС ЭВМ.— Минск : ИМ АН БССР, 1988.— С. 123—141.
25. *Prather R. E., Myers J. P.* The path prefix solftware testing strategy // IEEE Trans. Softw. Eng.— 1987.— 13, N 7.— P. 761—766.
26. *Ramamoorthy C. V., Ho Siu-Bun F., Chen W. T.* On the automated generation of program test data // Ibid.— 1976.— 2, N 4.— P. 293—300.
27. *Система* символического тестирования ПЛ-1-программ / Ю. В. Борзов, Г. Э. Дишлерс, И. Э. Медведис, Г. Б. Уртанс // УСиМ.— 1986.— № 6.— С. 75—79.



Поступила 29.06.88



Тел. для справок: 22-54-97 (Рига)

INFOTECH
STATE OF THE ART
REPORT

# SOFTWARE TESTING

## VOLUME 2:
## INVITED PAPERS

# SMOTL - A SYSTEM TO CONSTRUCT SAMPLES FOR DATA PROCESSING PROGRAM DEBUGGING

J Bicevskis
J Borzovs
U Straujums
A Zarins
Department of Computer Science
Latvian State University
E F Miller, Jr
Software Research Associates

*NOTE*

*This paper is an edited version of a version that has been submitted for publication in the IEEE Transactions on Software Engineering. The last author is responsible only for cleaning up the presentation and sharpening some of the language to minimize the chance of ambiguities; full responsibility for misinterpretation lies with the last author.*

# SMOTL - A SYSTEM TO CONSTRUCT SAMPLES FOR DATA PROCESSING PROGRAM DEBUGGING


## ABSTRACT

The possibility of automatic construction of a complete set of program tests is considered. A test set system is said to be complete if every feasible program branch (segment) is executed on it. The complete test set construction algorithm for commercially oriented data processing programs is outlined, and the results of its functioning on real programs are analysed.


Index terms: program validation, program testing, symbolic execution, test data generation, analysis of programs.

## INTRODUCTION

In recent years more and more attention has been paid to program validation. One of the common approaches is to test a program on different sets of input data (001,002, 003,004,005,006,008,009,010,011).

Although absolutely complete program testing can be achievable by executing it on all possible program inputs, most researchers agree that a natural criterion of program testing completeness is the execution of all program branches (or, in other words, traversing all exits of statements) for a finite number of cases (001,002,003,004, 008).

A path $P$ (a branch $P$) is called *feasible*, if there exists input data which forces the program to execute the path $P$ (the branch $P$). Further on we shall consider only those tests on which the program terminates normally. These are said to be *permissible*.

A finite test set $S$ for the given program will be called *a complete test set* (CSS), if it consists of permissible tests and every program branch that can be executed is executable on some test in $S$.

The idea of program debugging by CSS construction arises mainly from applications programming experience. The programmer always tries to select a set of tests such that the program executes all its branches. In this paper we shall discuss only the possibility of automatic CSS construction without any substantiation of this principle

The SMOTL system constructs a CSS automatically. It differs from other similar systems in the following two features:

1 SMOTL is oriented to commercially practical programs, whereas other known systems are meant for scientific programs

2 SMOTL is supplied with a strategy for choosing the program paths to analyse, whereas in the majority of other systems the task of selecting such paths is done manually. SMOTL constructs a CSS according to the program text without human interference.

## System survey

The SMOTL system is intended primarily for batch processing of programs written in SMOD; SMOD is a COBOL-like language, but has no means of direct access to data on the external storage devices.

Besides compiling and linkage editing, the SMOTL system contains facilities of the following kinds:

- Detection of certain run-time errors
- Constructing tests for program validation
- Putting out samples on the external storage devices
- Executing the program repeatedly on the constructed tests (regression testing)
- Printing and comparing program output data.

SMOTL is organized logically into six phases, described below:

1 Replacement of the source program by its internal representation in the form of a directed graph.

2 Static analysis of the program with the objective of time and memory space optimization by subsequent SMOTL phases.

3 Construction of a *covering set of paths*, i e, the test set containing only feasible paths and covering all feasible branches.

4 Minimization of the covering set.

5 Construction of test data for paths from a minimized covering set.

6 Output of test data to an external storage device, compiling and linkage editing of the SMOD text, repeated execution, and printing of generated results. This last phase has nothing to do with CSS construction and it will not be discussed in the paper.

## Problem solvability

We shall demonstrate below that for every real programming language the problem of CSS construction is solvable from the theoretical point of view. The crucial point to observe is that every program variable can be assigned only a finite number of different values. To construct a CSS we use an algorithm (described next) for exhaustive search of individual variables' values.

We begin the description of the algorithm by indicating how to check the execution feasibility of an individual path. Subsequently we shall discuss how the finite set

of all different paths can be treated.

We shall consider only initial paths in this section, i e, paths starting from the first statement. We execute statements of the path beginning from the first one. If the values of operands are known before the execution of a statement then the execution of this statement is carried out in the usual way by the simulation of program operation.

The values of operands may not be known in the following two cases:

1 When a data input statement is to be executed. In this area the corresponding values are to be assigned from input data, which certainly are not known. We evaluate these variables by assigning some concrete fixed values from the corresponding domains of possible values of the variables. These domains are uniquely determined by the specifications of the variables and this statement. It is important to note that these domains are finite because only numbers of limited length and precision, including strings of limited length, can be represented in the memory of a computer.

2 When there is a statement to be executed which does not read values from input data and nevertheless the values of some operands are not known. In this case the path is obviously infeasible and the further analysis is not needed.

The described process terminates in the following two cases:

1 When we have succeeded in executing a path from the beginning to the end. Consequently, there will be fixed values for all the input data. This fixation of the values determines the test data which forces the program to execute this path. Fixations of this kind are called *feasible fixations*. As soon as we have a sample which forces execution of a path, this path (obviously) is feasible.

2 When we have not succeeded in the path execution because of the fact that the selected fixation of input data values does not satisfy some conditional statement of the path. In this case we must repeat execution of the path, fixing another value, for input data. As soon as the domains of possible values of all variables are finite and every path contains only a finite number of data input statements, the number of all different possible fixations then also becomes finite. The feasibility of the path can be checked by effectively examining all of these fixations.

We have proved that the feasibility of any individual program path can be determined. Now we shall demonstrate how the infinite set of all different initial paths can be treated. We can examine the paths in ascending order of their length and try to select a set of feasible paths having the program exit statement for the last statement of the path (i e, a STOP path) which covers every program branch. CSS construction will be completed if we succeed in finding such a set.

However, if there is a program branch which has no permissible samples for execution of it then the described procedure does not terminate. This happens by having to consider paths of continually increasing length and waiting till some feasible STOP-path contains the chosen branch. Thus we are forced to analyse an infinite number of paths (except in the case of trivial programs without loops).

This apparent difficulty can be avoided, nevertheless. It is clear that at every instant the program's further behaviour is uniquely determined by the values of program variables

and does not depend on the path on which these values are assigned. More precisely: let feasible paths PA and PB end with the same statement and let the values of the variables at the end of the path PA, for a feasible fixation A, coincide with the values of the corresponding variables at the end of the path PB for a feasible fixation B. Then, if the path obtained by concatenation of path PA and its continuation PC is feasible, then so is the path obtained by concatenation of paths PB and PC. Note that if two paths differ only in the number of loop traversals and if the value of each variable at the loop exit statement is equal for both paths then there is no need to consider the longer path.

Hence, let us consider only paths with different values of variables at the exit statements of loops. The number of such paths is of course finite; this fact permits us to construct a CSS in all cases.

From the practical point of view, however, the algorithm described cannot be applied directly because of the possible need of an unending exhaustive search. To find more suitable methods for CSS construction we have to take into account that the domains of variables' values are practically infinite. In theoretical investigations (001, 002,004,010,011) the infinity of the domain is assumed explicitly. Under this assumption the CSS construction problem is theoretically unsolvable. However, the CSS construction algorithm discussed here applies to a wide class of practical data processing oriented programs.

The algorithm implemented in SMOTL is similar to the one described above, but instead of concrete values of variables a generalized description of possible values, called a concept 'state', is used. It is possible, by using some standard optimization methods, to reduce the number of different states to a small and manageable number for the case of real-world data processing oriented programs. This technique allows us to construct an acceptable CSS in realistic computation times.

## DETAILED SMOTL DESCRIPTION

### Phase I

The first phase of SMOTL operation replaces the source program by a directed graph. The nodes in the graph are the statements of the program and the edges represent the program flow. Every node in the graph contains not only the statement code but also references to the descriptions of operands. Such representation permits easy traversal of the program paths and compact storage of the program in the core. (About 4K bytes are needed for a program containing 300 statements, to cite a specific example).

In discussing the next phases of SMOTL operation we shall use PL/1-program EXAMPLE to demonstrate the system functioning. This program forms the file PAYBILL by processing the ordered files PAY and NAME (see Figure 1).

### Phase II

The reader may easily pass over the following definitions of essentially located statements (ELSs) and essential variables without influencing his understanding. One may consider all program statements as ELSs and all program variables as essential variables.

```
EXAMPLE: PROC OPTIONS (MAIN);
      DECLARE PAY FILE INPUT;
      DECLARE NAME FILE INPUT;
      DECLARE PAYBILL FILE OUTPUT;
      DECLARE 1 P,
                  2 CODEP PICTURE '(4)9',
                  2 INCOME PICTURE '(4)9V99',
                  2 CREDIT PICTURE '(4)9V99';
      DECLARE 1 N,
                  2 CODEN PICTURE '(4)9',
                  2 NAMEN PICTURE '(4)9V99';
      DECLARE 1 PRINTLINE,
                  2 CODE PICTURE '(4)9',
                  2 FILL1 CHARACTER (10) INITIAL ('  '),
                  2 NAMEP CHARACTER (12),
                  2 FILL2 CHARACTER (20) INITIAL ('  '),
                  2 OUTCOME PICTURE '(4)9V99';
L0: OPEN FILE (PAY), FILE (NAME), FILE (PAYBILL);
      ON ENDFILE (PAY) GOTO L9;
      ON ENDFILE (NAME) GOTO L6;
L1: READ FILE (PAY) INTO (P);
L2: READ FILE (NAME) INTO (N);
L3: IF CODEP < CODEN THEN DO; DISPLAY ('NAME NOT FOUND'); GOTO L8; END;
L4: IF CODEP > CODEN THEN GOTO L2;
L5: OUTCOME = INCOME - CREDIT; CODE = CODEP; NAMEP = NAMEN;
      WRITE FILE (PAYBILL) FROM (PRINTLINE); GOTO L8;
L6: CODEN = +9999; GOTO L3;
L7: CODEP = +9999;
L8: READ FILE (PAY) INTO (P); GOTO L3;
L9: CLOSE FILE (PAY), FILE (NAME), FILE (PAYBILL);
END EXAMPLE;
```

*Figure 1: Text of the program EXAMPLE*

For the sake of memory efficiency, however, we have tried to minimize the number of ELSs and essential variables.

The second phase of SMOTL operation selects a set of program statements such that every loop contains at least one statement from this set. We shall call the statement from this set *Essentially Located Statements* (ELS). Usually the ELS is a loop exit statement. In addition, the first program statement and every program EXIT statement are considered to be ELSs. If several loops have a common part, then the ELSs are selected from this common part in order to minimize the number of ELS. In the program EXAMPLE the statements labelled L0, L3 and L9 are the ELSs.

Associated with every ELS there is a list of variables we call *essential variables associated with the ELS*. A variable *x* is said to be an essential variable for certain ELS if there exists a path beginning with this ELS such that the following statements hold true:

1 The variable *x* is analysed in some conditional statement of this path, or the variable *x* enters some expression which is analysed in a conditional statement of this path.

2 The value the variable *x* has immediately before execution of the ELS is not changed until it is analysed in the conditional statement of the path.

For instance, in the program EXAMPLE the statement L0 has no essential variables because on every path which follows L0 every variable is assigned a new value from the input file before it is analysed in a conditional statement. L9 has no essential variables because there are no paths following this statement that conditional statement L3 has the essential variables CODEP and CODEN assigned to it.

Unreachable program statements can be detected along with selection of the ELSs; references to uninitialized variables can be found when identifying essential variables. EXAMPLE contains an unreachable statement L7 but it does not contain references to uninitialized variables.

Finally, the second phase of SMOTL produces a list of all the possible paths from one ELS to another (further referenced as subpaths). In EXAMPLE such subpaths are as follows:

(L0,L1,L2,L3),
(L0,L1,L2,L6,L3),
(L3,L8,L3),
(L3,L8,L9),
(L3,L4,L2,L3),
(L3,L4,L2,L6,L3),
(L3,L4,L5,L8,L3),
(L3,L4,L5,L8,L9),
(L0,L1,L9).

Phase III

The third phase tries to construct a set of feasible paths ending with the program exit statement such that this set contains every feasible branch. The most important remaining problem with the method used in SMOTL is to choose a criterion for procedure termination in the case when some program branch is infeasible.

This problem can be solved by the aid of a concept of the *state* - a generalized description of all possible values of a variable. Because of the fact that the construction of a state is highly complicated and depends on the features of concrete programming language we shall only illustrate the construction of a state by simple examples in this paper. States will be used for the following two purposes:

1 If there are two paths which differ only in number of loop traversals we shall not consider the longer path in the case when the states associated with the last statements of the paths coincide.

2 When we investigate the feasibility of a path which is a concatenation of paths PA and PB, the concept of the state is used for possibility of separate analysis of PA and PB. A state contains all the information that must be known in addition to the fact of feasibility of the path PA in order to recognize the feasibility of the path PB. The feasibility of the path PB can be recognized from the fact of feasibility of the path PA and from the set of all states ascribed to the last statement of PA. Moreover, it is guaranteed; if the paths PA and PB are feasible, so is the path obtained by concatenation of PA and PB.

The third system consists of STRATEGY and ANALYSER. STRATEGY chooses a subpath and passes it on to ANALYSER together with description of relationships of variables at the beginning of the subpath (i e, its state). ANALYSER'S task is to decide if the given subpath is feasible with the given state. If the subpath is feasible ANALYSER constructs a new state and STRATEGY ascribes it to the last statement of this subpath. Further on STRATEGY selects a new subpath and state and turns to ANALYSER again, etc. The end of the work of the third phase is determined by STRATEGY.

*STRATEGY*

STRATEGY begins its operation from the first program statement and the empty state. It selects a subpath to the adjacent ELS and turns to ANALYSER with the question: 'Is this subpath feasible with the empty state?'. If the subpath is feasible, then ANALYSER constructs a state which describes the values of essential variables after the execution of the analysed subpath. Then STRATEGY ascribes the new state to the ELS which is the last statement of the analysed subpath. Further STRATEGY selects another subpath which (a) follows the subpath investigated before, and (b) has not yet been analysed with the newly constructed state. ANALYSER investigates the selected subpath with the given state, etc. In that way STRATEGY constructs a feasible path adding one subpath to it on every step. If some subpath turns out to be infeasible then STRATEGY backs up one subpath back and selects another subpath to analyse.

The attempted construction of a feasible path in this manner terminates for one of the following two reasons:

1 The program exit statement is reached

2 STRATEGY comes to an ELS traversed earlier and the newly constructed state coincides with one of those ascribed to this ELS before and, in addition, every subpath following the ELS has already been investigated (with the same state) in some earlier step.

In the first case STRATEGY includes the constructed path into the covering set of paths. In the second case STRATEGY keeps the constructed path in mind. When construction of some other feasible path is completed STRATEGY examines whether it is possible to add some part of path ending with program exit statement to the prior path. If STRATEGY succeeds, the feasible STOP-path obtained will be included into the covering set.

After the construction of one path STRATEGY starts to construct a new feasible path. The starting point of the construction of the next path is an ELS on some path constructed before, chosen so that the number of non-investigated subpath/state pairs is the greatest possible. Here we make use of the second property of states, which allows the process to start the investigation of every path not necessarily from the first program statement, and which analyses every subpath with every state only once.

*ANALYSER*

ANALYSER's task is to decide if the given subpath is feasible with respect to the given state. ANALYSER performs a symbolic execution of a subpath. When a path is symbolically executed no values are assigned to the variables, as in the case of normal execution, but operations are performed on symbolic values. For instance, the READ statement usually assigns a value from the data set to a variable; in symbolic execution a variable receives a certain symbol for its value. This symbol denotes the location of the corresponding value in the data set. Thus after the path (L0,L1,L2,L3)

in the program EXAMPLE the variables CODEP and CODEN will receive the symbolic values $(PAY,1\text{-}4)_1$ and $(NAME,1\text{-}4)_1$, respectively. $(PAY,1\text{-}4)_1$ represents the value located in the first four bytes of the first record in the file PAY; $(NAME,1\text{-}4)_1$ represents the value located in the first four bytes of the first record in the file NAME. Symbolic execution of a path results in a system of inequalities which represents the constraints over symbolic values. By checking the consistency of the constraints (solving the system of inequalities) ANALYSER determines the path feasibility. In the given example the subpath (L0,L1,L2,L3) does not contain any conditional statement and the system of inequalities is empty. It has a solution; therefore the subpath (L0,L1,L2,L3) is feasible.

ANALYSER then constructs the state at the end of the subpath (L0,L1,L2,L3). A state is obtained by direct simplification of the inequality system. (Equalities expressing the values of essential variables at the end of the path are added automatically). In our example, the program state after the subpath (L0,L1,L2,L3) is the following:

$$CODEP = (PAY,1\text{-}4)_1$$
$$CODEN = (NAME,1\text{-}4)_1$$

On the next step STRATEGY turns to ANALYSER with the subpath (L3,L8,L3) and with the state constructed before. ANALYSER constructs the system of inequalities

$$(PAY,1\text{-}4)_1 < (NAME,1\text{-}4)_1,$$

which obviously has a solution. The newly constructed state is the following:

$$CODEP = (PAY,1\text{-}4)_2$$
$$CODEN = (NAME,1\text{-}4)_1.$$

Only one inequality is included because there are no essential variables having $(PAY,1\text{-}4)_1$ for their value. Hence $(PAY,1\text{-}4)_1$ cannot get new logical constraints and although new constraints on $(NAME,1\text{-}4)_1$ may appear the inequality $(PAY,1\text{-}4)_1 < (NAME,1\text{-}4)_1$ always will have a solution because of the practically unrestricted possibility to select the value for $(PAY,1\text{-}4)_1$.

The non-simplified system of inequalities could be used as a state but we would have too large a number of different states for a program having loops in this case. The main property of the simplified system of inequalities is the following: if both the given and the simplified systems (obtained for path PA) have some inequalities (obtained for an arbitrary continuation of PA) then they have solutions simultaneously, but the individual solutions may differ.

The further behaviour of the program depends only on relationships between the values of the variables. It does not depend on concrete values of the variables (in our case concrete symbolic values). Therefore we can introduce the following definition of coincidence of states.

Two states are said to coincide if the corresponding systems of equalities and inequalities are equal up to names of the symbolic values.

Obviously, in our example the newly constructed state coincides with the one constructed before. Therefore STRATEGY selects the subpath (L3,L8,L9) instead of the subpath (L3,L8,L3) for further investigation. This subpath will appear to be feasible; thus STRATEGY will finish the construction of the first feasible STOP-path (L0,L1,L2,L3; L8,L3;L8,L9).

In general, the construction of a state merely consists of simplification of the system of inequalities obtained by the symbolic execution of a path.

One of the permitted simplifications has already been demonstrated in the example. There are many analogous simplifications and their formal description is given in (001,004,011). Because of the high complexity of these descriptions we do not discuss this matter further in this paper.

In some cases the system of inequalities built by ANALYSER may turn out to be non-linear, in which case finding a solution is an unsolvable problem. Practical experience suggests that such non-linear systems seldom appear in data processing. A method of solving systems of inequalities was developed for the SMOTL system. This method does not guarantee the finding of a solution in all the cases even when it is known to exist. However, the method is very fast; the essence of the method is described next.

First, assume that every variable from the inequalities has been ascribed a segment of its possible values. This segment is determined by the specification of the variable and by the inequalities, in which the variable is compared with constants. For instance, if a variable is specified by the statement

```
DECLARE x DECIMAL FIXED (2);
```

and the system of inequalities contains the inequality $x > 13$, then the variable $x$ is contained in the segment $[14,99]$.

Recall that the system of inequalities is being repeatedly examined; for every inequality and for every variable in the inequality the segment corresponding to this variable is 'corrected'. For instance, if the inequality is $x < y$, the corresponding segments for variables $x$ and $y$ are $[a,b]$ and $[c,d]$, and $x$ and $y$ are declared to be integers, then after the correction the segments are $[a,\min(b,d-1)]$ and $[\max(a+1,c),d]$.

The procedure terminates when the current pass examining the inequalities does not change any segment or when some segment becomes empty. In the latter case the system of inequalities has no solution and therefore the analysis of it is finished. In the former case, if the system of inequalities contains no arithmetical expressions then the variables will have been assigned the lower bounds of the corresponding segments. The assigned values represent the solution of the system.

However, if the system of inequalities contains arithmetical expressions, then the solution is found by a restricted search of the values from the segments (the exhaustive search may take too much time). If ANALYSER does not succeed in this restricted search then it asserts (perhaps incorrectly) that the given subpath is infeasible. Consequently it cannot be guaranteed that the sample system constructed by SMOTL is complete in these cases.

*Additional remarks*

Every path which is recognized by STRATEGY to be feasible and which eventually reaches the program exit statement is added to the covering set of the path being constructed. When added, a check is made to determine whether every program branch is already represented in the set. If it is, the third phase terminates. However, if the program contains an infeasible branch, STRATEGY is forced to check all the possibilities, i e, every subpath with every state.

The third phase terminates in two more cases:

1 Lack of memory for storing of the newly constructed state (SMOTL can store approximately 200 states on average)

2 The time allowed for the work of the third phase is spent (typically, 5 minutes of CPU time is allowed for the third phase in SMOTL).

As mentioned earlier, in these two cases it is not guaranteed that the constructed test set is complete.

Analysing the program EXAMPLE, STRATEGY produces the following set of feasible paths:

(L0,L1,L2,L3;L8,L3;L8,L9)
(L0,L1,L2,L3;L4,L2,L3;L8,L9)
(L0,L1,L2,L3;L4,L2,L6,L3;L8,L3;L8,L9)
(L0,L1,L2,L3;L4,L5,L8,L3;L8,L9)
(L0,L1,L2,L3;L4,L5,L8,L9)
(L0,L1,L2,L6,L3;L4,L5,L8,L3;L8,L9)
(L0,L1,L2,L6,L3;L4,L5,L8,L9)
(L0,L1,L9).

Here a semi-colon separates the subpaths within each path.

Phase IV

In batch debugging it is not convenient to deal with a large number of tests. Therefore the fourth phase rearranges and rejects certain paths produced by the third phase. The aim is to reduce the number of paths in the covering set.

For instance, the paths (L0,L1,L2,L3;L8,L3;L8,L9) and (L0,L1,L2,L3;L4,L2,L3;L8,L9) can be merged into one path (L0,L1,L2,L3;L4,L2,L3;L8,L9). Here we make use of the fact that the program states ascribed to ELS L3 after traversing the paths (L0,L1,L2,L3), (L0,L1,L2,L3;L8,L3) and (L0,L1,L2,L3;L4,L2,L3) coincide. It is easy to check that the newly constructed path contains the same branches as the two given paths. Due to the properties of states the feasibility of the newly constructed path is guaranteed.

We shall obtain as a result for the program EXAMPLE a covering set of paths, which contains only these two paths:

(L0,L1,L2,L3;L8,L3;L4,L2,L3;L4,L5,L8,L3;L4,L2,L6,L3;L8,L3;L8,L9)

and

(L0,L1,L9).

Phase V

The fifth phase constructs test data values for the paths selected by the preceding phases. The only data obtained at analysis time contains information about a very few values of input records, because a significant portion of them have been rejected when the states were constructed. Therefore the symbolic execution of the entire path

is performed once more to obtain a system of inequalities in all-symbolic-values form. By solving these inequalities we obtain the values for the input which forces the program to execute the corresponding path.

For the program EXAMPLE the complete test set is as shown in Figure 2:

$T_1$

| file PAY | file NAME |
|---|---|
| 1-st record 0001111111000000 | 1-st record 0002 J. BORZOVS |
| 2-nd record 0003222222333333 | 2-nd record 0003 J. BICEVSKIS |
| 3-rd record 0004444444444444 | |
| 4-th record 0005000000000000 | |

$T_2$

| file PAY | file NAME |
|---|---|
| empty | empty |

*Figure 2: Complete test set for the program EXAMPLE*

IMPLEMENTATION NOTES

The SMOTL system has been implemented in 1974-76 on a Soviet computer 'MINSK-32' (160K bytes main storage, CPU speed approximately 50 000 op/sec) (007). Direct access devices were not used. The selection of this environment was the result of our aim to construct a practical test data generation system. The SMOTL system consists of 30 000 computer instructions.

CSS construction time usually does not exceed the program's compile time. We have analysed the results of the system working on 39 programs. They were selected at random from the software of several already-implemented management systems. It turned out that our system processed programs having fewer than 300 statements sufficiently well. The programs contained an average of 11 conditional statements. The CSS was constructed for 16 programs out of 25; for the rest the test data obtained covered about 74% of all the feasible paths.

The situation was considerably worse in the case of programs containing more than 300 statements (there were 14 such programs). Practically useful results were obtained only for 5 of them. The average amount of conditional statements in one program from this class was 66. Five programs contained from 400 to 500 statements, and 7 had more than 500 statements.

Time and storage resources have proved to be insufficient for the processing of very long programs. Our concept of 'state' proved to be insufficient in the same cases, when highly complicated means were employed in a program such as irregular references to array elements or complicated arithmetic expression calculations.

## CONCLUSIONS

The advantage of such systems as SMOTL lies in generalized program testing. In testing with the constructed test data, the system demonstrates the function of all program parts that can be executed without abnormal termination. The SMOTL system gives diagnostic messages explaining the reasons of infeasibility of all the remaining (untested) program parts. SMOTL is especially convenient in batch-processing mode, where normal program termination gives a programmer much more information than its abnormal stop.

The current SMOTL implementation shows that for many data-processing-style programs it is possible to construct CSS in acceptable time on widely-used computers. We hope to obtain similarly positive results in developing our system for PL/1.

---

## REFERENCES


001 BARZDIN J M, BICEVSKIS J J and KALNINSH A A
*Construction of complete sample system for testing correctness of programs*
Uce-nye zapiski Latv gos univ vol 210 pp 152-188 Riga (1974)

002 KALNINSH A A, BICEVSKIS J J and BARZDIN J M
*Solvable and unsolvable cases of the problem of construction of a complete sample system*
Uce-nye zapiski Latv gos univ vol 210 pp 188-206 Riga (1974)

003 MILLER E F and PAIGE M R
*Automatic generation of software test-cases*
Eurocomp Conf Proc 1974 pp 1-12 (1974)

004 BARZDIN J M, BICEVSKIS J J and KALNINSH A A
*Construction of complete sample system for correctness testing*
Mathematical Foundations of Computer Science pp 1-12 Springer Verlag Berlin (1975)

005 KING J S
*A new approach to program testing*
Proc 1975 Intl Conf on Reliable Software (April 1975)

006 HOWDEN W E
*Methodology for the generation of program test data*
IEEE Trans Comput vol C-24 pp 554-559 (May 1975)

007 ERSHOV A P
*A history of computing in the USSR*
Datamation vol 21 no 9 pp 80-88 (Sept 1975)

008 HOWDEN W E
*Symbolic testing and the DISSECT symbolic evaluation system*
Computer Science Technical Report no 11 Applied Physics and Information Science University of California San Diego (May 1976)

009 CLARKE L A
*A system to generate test data and symbolically execute programs*
IEEE Trans on Software Engineering
vol SE-2 (Sept 1976)

010 BARZDIN J M, BICEVSKIS J J and KALNINSH A A
*Automated construction of complete sample system for program testing*
1977 IFIP Congress Proc pp 57-62
(1977)

011 BICEVSKIS J J
*Automatic construction of sample systems*
'Programmirovanije' no 3 60-70
Moscow (1977)

УДК 681.3.06

# ТЕСТИРОВАНИЕ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ СИМВОЛИЧЕСКОГО ВЫПОЛНЕНИЯ

**Ю. В. Борзов**

**Рассматривается тестирование программ, основанное на их символическом выполнении. Дается обзор действующих автоматических систем тестирования программ. Обсуждаются проблемы их разработки.**

## ИЗ ИСТОРИИ АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ

Первыми шагами по пути автоматизации тестирования были попытки машинной генерации тестов с использованием описаний структуры входных данных. Это было осуществлено еще в 1962 г. Р. Л. Саудером [1], а затем развито Х. Т. Хиксом [2], К. В. Ханфордом [3] и др. Однако в этих случаях тесты только внешне (по своему формату) напоминали те данные, для обработки которых была составлена программа. Конкретные значения тестов обычно генерировались случайно, и поэтому не было гарантии в том, что сгенерированные тесты проходят все ветви программы. Практика показала, что такой подход находит хорошее применение при отладке трансляторов. Об этом свидетельствует также использование этого подхода в СССР в рамках RTK-технологии [4]. При программировании задач других типов подход широкого применения не нашел.

С другой стороны, велись работы по созданию систем, автоматизирующих отдельные функции тестирования. Разрабатывались системы сбора статистической информации о тестируемой программе (число выполненных операторов, максимальные и минимальные значения переменных, выполнение определенных отношений между переменными и т. п.) [5—12]. Некоторые системы могли автоматически определять условия, которым должны удовлетворять входные данные для того, чтобы программа выполнила заданный путь [5, 13]. Системы [14, 15] выдавали множество путей программы, в совокупности содержащих все ветви программы.

Вся эта информация могла быть полезно использована при составлении тестов, которые, однако, должны были вестись вручную.

Наконец, в середине 70-х годов несколькими коллективами независимо и практически одновременно были разработаны экспериментальные системы тестирования с использованием символического выполнения программ. Часть из них своей основной целью поставили полную автоматизацию генерации тестов [13, 16—24] (при этом тесты должны целенаправленно и достаточно полно проверять программу в противоположность случайно генерируемым тестам). Другие старались рассматривать работу программы не на одном конкретном тесте, а на целом их классе. Конечной целью такого выполнения является определение соответствия между классами входных и выходных данных [18, 25—27].

## СИМВОЛИЧЕСКОЕ ВЫПОЛНЕНИЕ ПРОГРАММЫ

Каждый язык программирования имеет свою семантику выполнения, описывающую: а) область определения программных переменных (обычно это — числа); б) допустимые операции над значениями переменных (сложение, вычитание и т. д. чисел); в) правила передачи управления при выполнении программы.

Представляется целесообразным также определить так называемую семантику символического выполнения, при котором в качестве результатов выдаются формулы над символическими значениями (можно сказать, что таким образом мы заменяем арифметические вычисления алгебраическими).

Символическое выполнение I. Путем назовем конечную последовательность операторов программы $a_1, a_2, a_3, \ldots, a_{k+1}$, если для каждого $i$ $(1 \leqslant i < k)$ формально возможна передача управления из $a_i$ к $a_{i+1}$.

Цель символического выполнения программы первого вида — определить по заданному пути программы, каким условиям должны удовлетворять входные данные, чтобы при выполнении программы на этих данных управление передавалось именно по этому пути.

Пример. Пусть дана программа

```
POWER: PROCEDURE(X,Y);          1
       Z = 1;                   2
       J = 1;                   3
  LAB: IF Y >= J THEN           4
       DO;Z=Z* X;               5
         J=J+1;                 6
         GO TO LAB; END;        7
         RETURN(Z);             8
END POWER;                      9
```

Рассмотрим, каким условиям должны удовлетворять входные данные, чтобы был выполнен путь 1, 2, 3, 4, 5, 6, 7, 4, 8? Можно проследить за этим, если записывать после выполнения каждого оператора значения переменных программы и отношения между ними (назовем это состоянием). Тогда начальное состояние представится нам как:

а) значения: $X = \alpha_1$, $Y = \alpha_2$, $Z$ и $J$ — неинициализированы;

б) отношения: нет.

Здесь $\alpha_1$ и $\alpha_2$ — символические обозначения входных данных. Ниже приведены состояния после выполнения операторов 2, 3, 4, 5, 6, 7, 4, 8:

2 — а) $X = \alpha_1$, $Y = \alpha_2$, $Z = 1$, $J$ — неинициал., б) нет

3 — а) $X = \alpha_1$, $Y = \alpha_2$, $Z = 1$, $J = 1$ б) нет

4 — а) $X = \alpha_1$, $Y = \alpha_2$, $Z = 1$, $J = 1$ б) $\alpha_2 \geqslant 1$

5 — а) $X = \alpha_1$, $Y = \alpha_2$, $Z = 1 * \alpha_1$, $J = 1$ б) $\alpha_2 \geqslant 1$

6 — а) $X = \alpha_1$, $Y = \alpha_2$, $Z = 1 * \alpha_1$, $J = 1 + 1$ б) $\alpha_2 \geqslant 1$

7 — а) $X = \alpha_1$, $Y = \alpha_2$, $Z = 1 * \alpha_1$, $J = 1 + 1$ б) $\alpha_2 \geqslant 1$

4 — а) $X = \alpha_1$, $Y = \alpha_2$, $Z = 1 * \alpha_1$, $J = 1 + 1$ б) $\alpha_2 \geqslant 1$, $\alpha_2 < 1 + 1$

8 — а) $X = \alpha_1$, $Y = \alpha_2$, $Z = 1 * \alpha_1$, $J = 1 + 1$ б) $\alpha_2 \geqslant 1$, $\alpha_2 < 1 + 1$

Состояние после выполнения оператора 8 содержит необходимые и достаточные условия для выполнения рассматриваемого пути. Эти условия описываются системой неравенств б). Видно, что если только $1 \leqslant \alpha_2 < 2$, то будет выполнен именно нами рассматриваемый путь. Если бы система б) не имела решения, это означало бы, что данный путь невыполним ни при каких входных данных.

Помимо главной задачи — определения условий, налагаемых на входные данные,— мы получили также некоторую полезную дополнительную информацию: 1) параметр процедуры $X$ не влияет на выполнение пути;

2) результатом процедуры при выполнении данного пути будет значение переменной $Z = \alpha_1$.

Одновременно получили возможность генерировать тест для проверки данного пути. Им будет пара чисел, где первое число произвольно, а второе есть любое решение системы неравенств б) в конце пути 1, 2, 3, 4, 5, 6, 7, 4, 8.

То, что символически выполняли заранее фиксированный путь начиная с его начала, не существенно. Ту же самую информацию можно получить также при подобном анализе начиная с конца пути (именно такой анализ исторически был предложен раньше [5, 13, 17]). Анализ с конца пути (trace backwards) позволяет вообще не рассматривать те переменные, которые не влияют на передачу управления. Таким образом, алгоритм является более быстрым, что существенно при практической реализации.

Правда, можно уже заранее определить те переменные в программе, которые на передачи управления не влияют, и затем при символическом выполнении их не учитывать. Именно так обычно и поступают, если применяется выполнение с начала пути (trace forwards) [23]. Этот способ символического выполнения имеет то преимущество, что невыполнимость пути может быть установлена в месте возникновения ошибки (обращение к неинициализированной переменной, отсутствие решения системы неравенств б) и т. п.), не доходя до конца пути, что невозможно в случае выполнения с конца пути.

Символическое выполнение II. Целью символического выполнения программы второго вида является определение по зафиксированным ограничениям на символические входные данные пути программы, который будет выполнен на этих данных, и символических значений переменных в конце пути.

Конечной целью (но не всегда достижимой) при этом является получение отображения входных данных в выходные данные, т. е. функции, которую реализует программа. В качестве примера рассмотрим ту же процедуру POWER.

Обозначим, как и в предыдущем случае, значение первого параметра через $\alpha_1$, второго — через $\alpha_2$. Допустим, что задано следующее ограничение на входные значения: $\alpha_2 < 1$.

Нетрудно установить, что состояние переменных после выполнения операторов 1, 2, 3 будет

(1) а) $X = \alpha_1$, $Y = \alpha_2$, $Z = 1$, $J = 1$, б) $\alpha_2 < 1$.

Это состояние позволяет нам определить, что условие $Y >= J$ ложно (так как не может быть $\alpha_2 \geqslant 1$ из-за (1)) и, следовательно, следующим будет выполнен оператор 8. Результатом работы процедуры будет значение $Z = 1$.

Очевидно, таким образом удалось установить, что в случае показателя степени меньшего единицы (значение второго параметра процедуры), возведение в степень запрограммировано ошибочно. Тут полезно указать, что одиночный тест с начальным значением $Y = 0$ не обнаружил бы ошибки, хотя и это значение взято из только что рассмотренной области входных значений.

Следует, однако, что не во всех случаях можно определить, по которой из потенциально возможных ветвей программы надлежит продолжать символическое выполнение. Например, если в рассмотренном выше примере заменить ограничение на $\alpha_2 < 2$ (или вовсе снять ограничение), то определить путь дальнейшего выполнения, начиная с оператора 4, не удастся, так как возможны обе альтернативы (при $1 \leqslant \alpha_2 < 2$ управление передается на оператор 5, а при $\alpha_2 < 1$ — на оператор 8). В таком случае обычно исследуются обе альтернативы, т. е. процесс символического выполнения разветвляется. Каким образом установить, по какому пути продолжать символическое выполнение в случае разветвления (IF-оператор)? Обычно

используется один из двух способов. Первый из них заключается в следующем: берем систему отношений над символическими входными значениями из состояния переменных перед выполнением IF-оператора (обозначим эту систему через $\alpha$). Добавляем к $\alpha$ булевское выражение из IF-оператора, в котором на место переменных проставлены соответствующие символические значения (это выражение обозначим через $\beta$). Полученная система $\alpha + \beta$ за редкими исключениями, которыми здесь пренебрежем, является системой равенств и неравенств над символическими входными значениями. Одновременно составляем и систему $\alpha + \neg\beta$. Решаем обе системы.

Если система $\alpha$ имеет решение (т. е. существуют входные данные, на которых программа выполнится до рассматриваемого IF-оператора), то возможны три случая:

1) $\alpha + \beta$ имеет решение, а $\alpha + \neg\beta$ решения не имеет;
2) $\alpha + \beta$ не имеет решения, а $\alpha + \neg\beta$ имеет решение;
3) как $\alpha + \beta$, так и $\alpha + \neg\beta$ имеет решение.

В случае 1) следует продолжать символическое выполнение по THEN-ветви, в случае 2) — по ELSE-ветви, а в случае 3) процесс выполнения должен содержать продолжение по обоим ветвям (разветвление процесса символического выполнения).

Второй способ определяется формальным доказательством теорем.

Обозначим через pc (path condition [25]) конъюнкцию отношений над символическими входными данными из состояния переменных перед выполнением IF-оператора.

Параллельно пытаемся доказать две теоремы: 1) $pc \supset \beta$ и 2) $pc \supset \neg\beta$. Самое большее, лишь одна из этих теорем может оказаться истинной (исключая тривиальный случай, когда *pc* тождественно ложно). Если истинна теорема 1), то символическое выполнение следует продолжать по THEN-ветви; если истинна 2), то — по ELSE-ветви. Если ни 1), ни 2) не являются тождественно истинными выражениями, то должно произойти разветвление процесса символического выполнения (т. е. рассматриваются обе альтернативы).

Заканчивая описание обоих способов, подчеркнем, что разветвление символического выполнения в обоих случаях связано с конкретным выполнением IF-оператора на определенном пути и не связано с самим IF-оператором. Выполнение IF-оператора на одном пути может породить разветвление процесса символического выполнения, а следующее выполнение того же оператора на другом пути — нет.

## ПРЕИМУЩЕСТВА СИМВОЛИЧЕСКОГО ВЫПОЛНЕНИЯ

Применение символического выполнения имеет следующие основные преимущества.

1. Символическое выполнение первого вида, которое применяется для генерации тестов, позволяет образовать тесты целенаправленно, учитывая логическую структуру программы. Таким образом, удается всесторонне проверить программу, например, автоматически составить тесты так, чтобы на них были выполнены все ветви программы. Подобное составление тестов невозможно при случайной их генерации.

2. Символическое выполнение второго вида дает возможность одним выполнением программы на символических входных данных заместить огромное (иногда бесконечное) число выполнений на конкретных значениях. Если при этом удается разбить область входных значений программы на классы, где каждый класс содержит конкретные значения, эквивалентные с точки зрения логики программы, то выполнение программы на символических значениях, представляющих эти классы, нередко оказывается полным (и конечным) перебором всех возможных входных значений.

Еще одной особенностью символического выполнения надо считать то, что выходные значения представляют собой выражения над символически-

ми входными значениями, т. е. получаем более или менее исчерпывающее описание функции, которую реализует программа. При выполнении программы на конкретных значениях часто практически невозможно установить соответствие между входными и выходными значениями, т. е. почти невозможно по конечному множеству пар (входное значение, выходное значение) определить функцию, реализуемую программой.

## СИСТЕМЫ СИМВОЛИЧЕСКОГО ВЫПОЛНЕНИЯ ПРОГРАММ И АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ ТЕСТОВ

В середине 70-х годов разработано около десятка систем, базирующихся на символическом выполнении программ. Часть из них в качестве основной функции имеют генерацию тестов [17, 19—24], а некоторые — само символическое выполнение программы [18, 25, 27]. Первые при этом используют символическое выполнение первого вида для образования систем равенств и неравенств в символических входных значениях (путем решения систем затем генерируются тесты), а вторые используют символическое выполнение второго вида (в основном для получения символических выражений в качестве значений выходных переменных, т. е. по существу — отображения входных значений в выходные). И те и другие системы в процессе символического выполнения программы обнаруживают ряд динамических ошибок. Некоторые системы [18, 25, 27] дополнительно имеют средства проверки утверждений о программе.

Данные о наиболее известных системах символического выполнения программ и автоматической генерации тестов сведены в таблицу.

Две графы таблицы, по-видимому, требуют более детального объяснения. В графе 6 отмечены системы, которые в качестве результата символического выполнения выдают пользователю выражения (формулы) над символическими входными данными. В графе 9 отмечены системы, имеющие возможности проверки утверждений. Утверждениями обычно являются отношения между переменными программы, записанные в обозначениях типа Флойда — Хоара, ассоциированные с определенными точками в теле программы. Когда управление (будь это символическое или нормальное выполнение программы) передается на эти точки, система автоматически проверяет выполнение утверждения. Наиболее совершенной в этом смысле является система SELECT [18].

## ПРОБЛЕМЫ РЕАЛИЗАЦИИ

Применение подхода символического выполнения для реальных языков программирования сталкивается с рядом проблем, которые являются общими для всех или почти всех разработчиков систем, базирующихся на этом подходе.

1. Число путей программы, как правило, очень велико. При этом их длина почти всегда ничем не ограничена. Как выбрать пути для анализа, чтобы эти пути достаточно хорошо представляли программу? Как определить момент, когда прекратить анализ отдельного пути, если его длина не известна?

Большинство разработчиков эти вопросы решают наипростейшим образом одним из следующих способов: а) пользователь заранее задает полностью весь анализируемый путь; б) пользователь заранее задает максимальную длину анализируемых путей или максимальное число прохождения циклов; в) пользователь в диалоговом режиме сам выбирает путь для анализа и пооператорно выполняет его.

Так поступают авторы систем EFFIGY [25], SELECT [18], DISSECT [27] и Л. Кларк [20]. Лишь авторы систем CASEGEN [21] и СМОТЛ [28] явно не ограничивают ни число, ни длину анализируемых путей. Однако первые формально выбирают пути все большей длины до тех пор, пока покрывают-

| Название системы | Авторы | Место разработки | Пакетная или диалоговая | Вид символической интерпретации | Есть ли выдача формул | Генерация тестов | Входной язык | Проверка утверждений | Год первой публикации | Метод выбора анализируемых путей | Примечания |
|---|---|---|---|---|---|---|---|---|---|---|---|
| EFFIGY | J. C. King<br>S. M. Chase<br>A. C. Chibib<br>J. A. Darringer<br>S. L. Hantler | U.S.A.<br>New York<br>Yorktown Heights<br>IBM T. J. Watson Research Center | Д | II | + | — | Подмножество PL/1 (целочисленные переменные и одномерные массивы) | + | 1975 | Задаются пользователем | |
| SELECT | R. S. Boyer<br>B. Elspas<br>K. N. Levitt | U.S.A.<br>California<br>Menlo Park<br>Stanford Research Institute | Д или П | II и I | + | + | Подмножество Lisp | + | 1975 | Задаются пользователем или выбираются автоматически по заданной максимальной длине путей | |
| | L. A. Clarke | U.S.A.<br>Massachusets<br>Amherst<br>University of Massachusets | Д или П | I | + | + | Ansi Fortran | — | 1975 | Задаются пользователем | Основная функция—генерация тестов |
| | W. E. Howden | U.S.A.<br>California<br>La Jolla<br>University of California | П | I | — | + | Fortran | — | 1975 | Могут выбираться автоматически | Система не закончена (разработаны лишь 2 первых блока из 5). |
| DISSECT | W. E. Howden<br>F. Laub<br>R. E. Hoffman | U.S.A.<br>California<br>La Jolla<br>University of California | П | II | + | — | Fortran | + | 1976 | Задаются пользователем или выбираются автоматически, если задано максимальное число прохождения циклов | |
| | C. V. Ramamoorthy<br>Siu-Bun F. Ho | U.S.A.<br>California | П | I | — | + | Fortran | — | 1976 | Выбираются автоматически | Система является экспериментальной |

| Название системы | Авторы | Место разработки | Пакетная или диалоговая | Вид символической интерпретации | Есть ли выдача формул | Генерация тестов | Входной язык | Проверка утверждений | Год первой публикации | Метод выбора анализируемых путей | Примечания |
|---|---|---|---|---|---|---|---|---|---|---|---|
| EFFIGY | J. C. King<br>S. M. Chase<br>A. C. Chibib<br>J. A. Darringer<br>S. L. Hantler | U.S.A.<br>New York<br>Yorktown Heights<br>IBM T. J. Watson Research Center | Д | II | + | — | Подмножество PL/1 (целочисленные переменные и одномерные массивы) | + | 1975 | Задаются пользователем | |
| SELECT | R. S. Boyer<br>B. Elspas<br>K. N. Levitt | U.S.A.<br>California<br>Menlo Park<br>Stanford Research Institute | Д или П | II и I | + | + | Подмножество Lisp | + | 1975 | Задаются пользователем или выбираются автоматически по заданной максимальной длине путей | |
| | L. A. Clarke | U.S.A.<br>Massachusets<br>Amherst<br>University of Massachusets | Д или П | I | + | + | Ansi Fortran | — | 1975 | Задаются пользователем | Основная функция—генерация тестов |
| | W. E. Howden | U.S.A.<br>California<br>La Jolla<br>University of California | П | I | — | + | Fortran | — | 1975 | Могут выбираться автоматически | Система не закончена (разработаны лишь 2 первых блока из 5). |
| DISSECT | W. E. Howden<br>F. Laub<br>R. E. Hoffman | U.S.A.<br>California<br>La Jolla<br>University of California | П | II | + | — | Fortran | + | 1976 | Задаются пользователем или выбираются автоматически, если задано максимальное число прохождения циклов | |
| | C. V. Ramamoorthy<br>Siu-Bun F. Ho | U.S.A.<br>California | П | I | — | + | Fortran | — | 1976 | Выбираются автоматически | Система является экспериментальной |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| CASE-GEN, СМОТЛ | C. V. Ramamoorthy Siu-Bun F. Ho W. T. Chen Y. W. Han | U.S.A. California Berkeley University of California | П | 1 | — | + | Fortran | 1976 | Выбираются автоматически | Система является экспериментальной подсистемой системы FACES |
| | Я. Я. Бичевский Ю. В. Борзов М. П. Василевский А. К. Зариньш У. М. Страуюмс | СССР, Рига, ВЦ Латвийского госуниверситета | П | 1 | — | + | Смод | 1977 | То же | Является единственной из перечисленных систем, которая ориентирована на задачи обработки данных |

ся все ветви программы. Поскольку не все из таким образом выбранных путей оказываются выполнимыми, то выбор циклически продолжается до тех пор, пока все ветви покрыты. Так как это не всегда возможно (в случае невыполнимости некоторой ветви программы), то такая процедура, вообще говоря, не всегда заканчивается.

В СМОТЛ анализ отдельного пути обрывается, если 1) в двух различных точках пути при одной и той же команде программы состояния переменных программы совпадают (из чего следует, что далее группы команд пути будут циклически повторяться) или 2) если путь проанализирован до конца или обнаружена ошибка. Анализ же всей программы заканчивается, когда или получено покрытие всех ветвей программы, или по одному разу пройдены все ветви программы при всех возможных состояниях и обнаружено, что из-за невыполнимости некоторой ветви программы полное покрытие получить в принципе невозможно.

2. Как на практике определить выполнимость пути программы? Из рассмотрения символического выполнения уже известно, что возможны два варианта — доказательство теорем или решение систем неравенств. На практике доказательство теорем используется только в системе EFFIGY [25]. Также не секрет, что имеющиеся на сегодняшний день машинные программы доказательства теорем далеки от совершенства. Поэтому подавляющее большинство рассматриваемых систем используют различные методы решения систем неравенств. Этот подход выгодно отличается еще и тем, что одновременно с определением выполнимости пути можно генерировать тест, на котором программа пройдет этот путь.

В большинстве случаев используются алгоритмы линейного программирования, что, конечно, сужает класс неравенств, которые можно таким образом решить. Так в SELECT сначала были испробованы алгоритмы линейного программирования GOMORY [29] и BENDERS [30], а затем разработчики перешли на использование алгоритма изменяемого градиента, который способен решать широкий класс систем неравенств, но зато требует диалога с человеком [18]. Л. Кларк в своей системе пользуется алгоритмом линейного программирования Ф. Гловера [20]. В. Миллер и Д. Спунер предлагают использовать для генерации тестов методы численной максимизации [31].

Следует отметить и два существенно отличных подхода. В системе CASEGEN применяется метод проб и ошибок с использова-

нием датчика случайных чисел [21]. В СМОТЛ работает быстрый алгоритм метода сегментов [28], дающий точные результаты в случае, когда значения переменных, влияющих на передачи управления, не подвергаются арифметическим преобразованиям. В противном случае производится ограниченный перебор значений, что не всегда заканчивается успешно. Однако, такая ситуация в программах типа обработки данных, на которые ориентирована СМОТЛ, встречается сравнительно редко.

Следует обратить внимание на то, что хороший и быстрый алгоритм решения систем неравенств во многом определяет успех применения рассматриваемых систем. Поэтому любые практические улучшения в этой области воспринимаются с огромным интересом.

3. Как обрабатывать индексированные переменные? Чтобы продемонстрировать ими порождаемую проблему, рассмотрим в качестве примера предикат: $a[i+1] = a[j]$. Этот предикат может быть удовлетворен, если положить $i + 1 = j$ или присвоить одно и то же значение обеим переменным $a[i+1]$ и $a[j]$. Если $i$ и $j$ получают в качестве значений входные данные, то эту неопределенность разрешить почти всегда невозможно. Рассмотрение же всех потенциально возможных случаев наталкивается на практически непреодолимый барьер огромного числа вариантов.

Наиболее простое решение состоит в обращении с вопросом к пользователю. Разработчики CASEGEN нашли другой интересный эвристический подход, заключающийся в том, что вначале решается система неравенств и равенств только для индексов индексированных переменных, после чего дальнейшее решение тривиально. В системе СМОТЛ производится слежение только за первыми четырьмя значениями индексированной переменной, предполагая, что в задачах типа обработки данных нерегулярное обращение к элементам массива крайне редко. Надо отметить, что удовлетворительное решение этой проблемы пока не найдено.

4. Как обработать обращения к функциям и подпрограммам? В большинстве рассматриваемых систем такой возможности нет. В тех случаях, где пытаются ее решить (например, в SELECT), это делается простой вставкой текста подпрограммы в тело анализируемой программы, что никак нельзя признать удовлетворительным, так как при этом увеличивается длина и сложность анализируемой программы и тем самым затрудняется ее анализ.

5. Как учитывать окончание машинных операций (в частности, арифметических)? Этот вопрос пока практически игнорируется.

Мы назвали только часть проблем реализации систем, основанных на символическом выполнении программ.

* * *

Нами были рассмотрены системы тестирования программ, базирующиеся на символическом выполнении.

В настоящее время многие коллективы трудятся над разработкой описанных систем. Можно надеяться, что на рубеже 80-х годов появятся промышленные системы символического выполнения программ и автоматической генерации тестов. По-видимому, они могут составлять одну из самых интеллектуальных частей систем автоматизации программирования и отладки. Работы в этом направлении важны для повышения эффективности программирования и надежности программ.

*Поступила в редакцию 13.XI.1978*

## ЛИТЕРАТУРА


1. *R. L. Sauder.* General Test Data Generator For Cobol.— AFIPS Conference Proceedings SICC, 1962, p. 317.
2. *H. T. Hicks.* The Air Force Cobol Compiler Validation System.— Datamation, 1969, *15*, No. 8, 73.
3. *K. V. Hanford.* Automatic Generation of Test Cases.— IBM Systems J., 1970, *9*, No. 4.
4. *И. В. Вельбицкий, В. Н. Ходаковский, Л. И. Шолмов.* РТК-технологический комплекс программиста БЭСМ-6.— Управляющие системы и машины, 1976, № 4, 29.
5. *R. M. Balzer.* EXDAMS — Extendable debugging and monitoring system. 1969 Spring Joint Computer Conference, AFIPS Conference Proceedings, Montvale, N. J., AFIPS Press., 1969.
6. *J. R. Brown, A. J. DeSalvio, D. E. Heine, J. G. Purdy.* Automated software quality assurance. Program Test Methods, N. Y.: Prentice-Hall, 1972, p. 76.
7. *L. G. Stucki.* Automatic generation of self-metric software. Proceedings 1973 IEEE Symposium on Software Reliability, p. 94.
8. *C. V. Ramamoorthy, S.-B. F. Ho.* Testing Large Software with Automated Software Evaluation Systems.— IEEE Transactions on Software Engineering, 1975, *SE-1*, 46.
9. *R. E. Fairley.* An Experimental Program-Testing Facility.— IEEE Transactions on Software Engineering, 1975, *SE-1*, No. 4, 350.
10. *C. V. Ramamoorthy, R. E. Meeker, J. Turner.* Design and construction of an automated software evaluation system. Proc. 1973 IEEE Symp. Software Reliability.
11. *В. В. Липаев, Л. А. Серебровский, В. В. Филиппович.* Система автоматизации программирования и отладки комплексов программ управления ЯУЗА-6.— Программирование, 1977, № 3, 87.
12. *О. Е. Мелконян.* Система отладки программ СЕКОНТ.— Программирование, 1977, № 3, 83.
13. *J. C. Huang.* An Approach to Program Testing. Computing Surveys, 1975, *7*, No. 3, 113.
14. *E. F. Miller, M. R. Paige.* Automatic Generation of Software Testcases. Eurocomp Conference Proceedings, 1974, p. 1.
15. *K. W. Krause, R. W. Smith, M. A. Goodwin.* Optimal software test planning through automated network analysis, in Rec. 1973 IEEE Symp. Computer Software Reliability.
16. *Я. М. Барздинь, Я. Я. Бичевский, А. А. Калниньш.* Построение полной системы примеров для проверки корректности программ.— В сб.: Уч. зап. Латв. гос. ун-та. Рига, 1974, т. 210, с. 152.
17. *W. E. Howden.* Methodology for the Generation of Program Test Data.—IEEE Transactions on Computers, 1975, *C-24*, No. 5, 554.
18. *R. S. Boyer, B. Elspas, K. N. Levitt.* SELECT — a Formal System for Testing and Debugging Programs by Symbolic Execution. Proceedings 1975 International Conference on Reliable Software, p. 234.
19. *E. F. Miller, R. A. Melton.* Automated Generation of Testcase Datasets. Proceedings 1975 International Conference on Reliable Software, p. 51.
20. *L. A. Clarke.* A system to Generate Test Data and Symbolically Execute Programs. IEEE Transactions on Software Engineering, 1976, *SE-2*, No. 3, 215.
21. *C. V. Ramamoorthy, S. B.— F. Ho, W. T. Chen.* On the Automated Generation of Program Test Data.— IEEE Transactions on Software Engineering, 1976, *SE-2*, No. 4, 293.
22. *R. H. Hoffman.* User information for the interactive automated test data generator. NASA Johnson Space Center, JSC Internal Note No. 75-FM-88, January, 1976.
23. *Я. Я. Бичевский.* Автоматическое построение систем примеров.— Программирование, 1977, № 3, 60.
24. *П. А. Правильщиков.* Построение тестов для программ.— Автоматика и телемеханика, 1977, № 5, 147.
25. *J. C. King.* A New Approach to Program Testing. Proceedings 1975 International Conference on Reliable Software, p. 228.
26. *J. C. King.* Symbolic Execution and Program Testing.— Communications ACM, 1976, *19*, No. 7, 385.
27. *W. E. Howden.* Symbolic Testing and the DISSECT Symbolic Evaluation System. Computer Science Technical Report No. 11, Applied Physics and Information Science, University of California, San Diego, La Jolla, 1976.
28. *J. Bicevskis, J. Borzovs, U. Straujums, A. Zarins, E. F. Miller.* Jr. SMOTL — a System to Construct Samples for Data Processing Program Debugging. Technical Note RN-415, Software Research Associates, P. O. Box 2432, San Francisco, CA 94126, April, 1978.
29. *R. E. Gomory.* An Algorithm for Integer Solutions to Linear Programs. Recent Advances in Mathematical Programming, R. L. Graves, P. Wolfe (eds.), McGraw-Hill, N. Y., 1963.
30. *J. F. Benders.* Partitioning Procedures for Solving Mixed-Variables Programming Problems. Numerische Mathematik 4, 1962, 238.
31. *W. Miller, D. L. Spooner.* Automatic Generation of Floating-Point Test Data.— IEEE Transactions on Software Engineering, 1976, *SE-2*, No. 3, 223.

# CONFERENCE PROCEEDINGS

## Eighth International Software Quality Week 1995

## 30 May – 2 June 1995

*The Sheraton Palace Hotel*
2 New Montgomery Street
San Francisco, California 94105





**Software Research Institute**
625 Third Street
San Francisco, CA 94107-1997 USA

Phone: (415) 957-1441 – FAX: (415) 957-0730

Paper 5-T-1

# Regression Testing of Software System Specifications and Computer Programs

*Ms. Zane Bicevska, Janis Bicevskis & J. Borzovs*
University of Latvia and
Riga Institute of Information Technology

**Mr. Zane Bicevska** was born in 1971. She received her Bachelor's degree in Computer Science from the University of Latvia in 1993. Currently she is a master student at the University of Latvia and is the principal programmer of the TW test tool and her interests include software testing.

**Mr. Janis Bicevskis** was born in 1944 and Graduated from the University of Latvia in 1971. He received his Ph.D. in Computer Science from the University of Latvia in 1992 and is now the principal developer of the SMOTL automated test generation system. He has written over 30 scientific papers and his interests include software testing, automated testcase generation and system development.

**Mr. Juris Borzovs** was born in 1950 in Finland. Graduated from the University of Latvia in 1973. He received his Doctorate in Computer Science from the University of Latvia in 1992. He is a member of IFAC, a reviewer for TCSE of the IEEE Computer Society and is Chairman of the Software Quality Section of the Latvian National Association for Quality. He has written over 50 published papers. His interests include software quality, software testing, programming methodology, software copyright, and terminology.

# Regression Testing of Software System Specifications and Computer Programs

**Z.Bicevska, J.Bicevskis, J.Borzovs**
University of Latvia and Riga Institute of Information Technology (Riga, Latvia)
Skanstes iela 13
LV-1013 Riga
Republic of Latvia
voice: (371) 2 377620
fax: (371) 782 1457
e-mail: jurisb@swh.lv

***Automated regression testing of information system specifications and computer programs is considered. Testing criteria differing from the traditional C1 are introduced - user interface (UI), data base (DB) and control flow (CF) criteria. The actual testing ideas implemented in the specifications test tool TW (Testing Workplace) are briefly described. Some possible ways to apply the mentioned ideas to computer programs are discussed.***

## 1. Software system specification testing

CASE-technology spreads widely into development of software systems. Previously used informal system descriptions are replaced by strictly formalised specifications. These specifications are subjects of translation into executable code by means of corresponding CASE-tools. So the programmers work is usually saved, although our principal interest using CASE-tool lies in obtaining formalised specifications of software system to be developed. Different from procedures that were developed in COBOL or other programming languages in former times, the obtained specifications operate on such rather new concepts as date base ER-model, data flow diagram, SQL query, menu, screen and report forms, etc.

Two questions can be raised: whether there is a need for activity similar to program testing and what can be understood by specification testing in general?

Obviously, there is no need, except a few cases, to check CASE-tool generator being used to create executable code of software system. By any case we have to rely upon thesis that system specifications translated into corresponding executable code are correct, and user should not test CASE-tool software. Still, there is an actual need for an activity that proves that the developed system does exactly what it has to do and what the developer wants it to do. In fact, only the description level of system under development has changed, namely, less dynamic but informatively more concentrated object - system specification - substitutes programs. The very essence of testing remains - to become convinced of correct operation of the developed system by means of carefully selected testcases.

Due to the fact that CASE-technology allows to obtain a prototype of operational system relatively easy the idea of prototype creation for a software system to be developed becomes more and more popular. Using such a prototype the inconsistency between user requirements and developer assumptions can be solved at an early stage of system development. Hence the system prototype can be many times redeveloped, refined step by step towards a customer desired system, and finally transformed into the target software system. Like any functional (and executable) description of a system to be developed, the prototype also can be verified, i.e. tested. At the same time each current version of prototype should possess many features of the former one. This can be checked by means of regression testing.. Usefulness of testing of information system specifications during prototyping phase is substantiated by given error statistics [1]. Almost half of all detected errors were caused by incorrectness in the system specifications made during its development period.

## 2. Transition from programs to specifications

This paper offers to apply traditional testing methods, e.g.[ 2], in specification testing. The popular program test criteria based on control flow graph are to be substituted by the ones ensuring test of interoperability between different specification elements, e.g. checking of various cases of data base filling and usage, screen form filling, reactions to different mouse or keyboard actions, etc. At the same time, the fundamental test principles are preserved:

- A test model is assigned to an object (program or specification) to be tested.
- Testcases to be applied to the test object are selected adhering to the test model.
- Test process is considered complete (finished) when the test object correctly operates on a set of tests that sufficiently covers the test model.

The most often a test model is represented in form of graph. Vertices represent features of system under test. Edges represent testable operations or relationships between features. In our case, to cover sufficiently a test model indeed means to cover a test model graph. We propose to use the following two criteria along with specification test models.

UI criterion. Vertices represent windows of system to be tested, edges - feasible actions on them, e.g. pushing of ENTER key, OK button, mouse click, etc. In this case test process is considered complete when the tested system correctly operates on such set of tests that contains each of the expected feasible windowing operation of the system.

DB criterion. Vertices represent windows and data bases of tested system, edges - feasible data base operations from window menus, e.g. file read, store, etc. Test process is considered complete when the tested system correctly operates on such set of tests that contains each feasible data base operation.

## 3. The TW test tool

The remaining part of this paper contains description of the system TW which embodies the abovementioned ideas. The TW is a test tool oriented to testing of specifications written in the specification language GRAPES [3]. The TW provides a regression testing of information system specifications which are developed by independent system designing and prototyping tool. Informally, by *specification* we mean a description of information system structure, functioning, data structures and user interface. The structure of the system under test is described like data flow diagram; data structure is represented by ER-model and corresponding data structures of entities; user interface - by screen and report forms; system functioning - by procedures.

Although the TW has been originated in GRAPES/GRADE environment [4], due to its well-modularised architecture and flexible interfacing it can be rather easily ported to other test environments.

The rest of the paper does not deal specifically with GRAPES/GRADE because the reader could well be not very familiar with it. Therefore examples of UI and DB test models in the next section will be applied to the widely used word processor WORD.

The TW provides gradual testing, the first three stages of which cover test suite collection process but the last two ones - allow to perform regression testing process:

- building of several system coverage models - CF, DB, DU, UI - thus showing the structure of system from different viewpoints,
- collecting of test suites through execution of system prototype,
- setting of check (correct by definition, resulting) values,
- executing of system prototype with the collected test suites and comparing of actually obtained values with stored check values,
- evaluation of test results.

## 4. Examples

In this section we demonstrate several simple examples to support the previously expressed information. As we mentioned earlier they relate to word processor WORD. The first example (Fig. 1) represents a fragment of window test model (entire model is too great), the second one (Fig. 2) represents again a fragment of file management model that ensures retrieval and storage of text from/into files, and the third one (Fig. 3) is a composition of the both. In fact, the system TW is able to obtain automatically similar models from description of system under test when the latter is described by means of the GRAPES specification language.

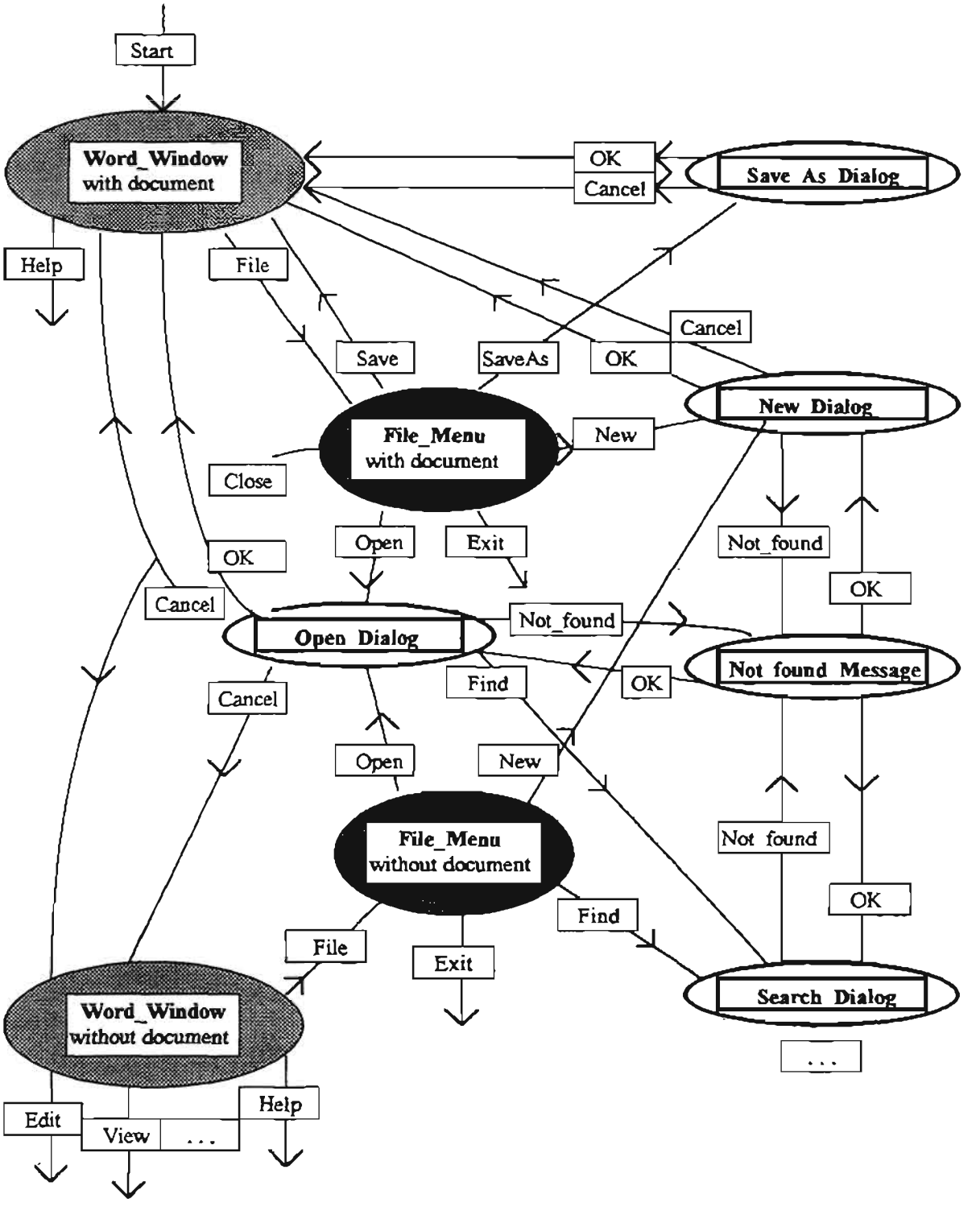



Fig 1. User interface test model fragment for MS WORD

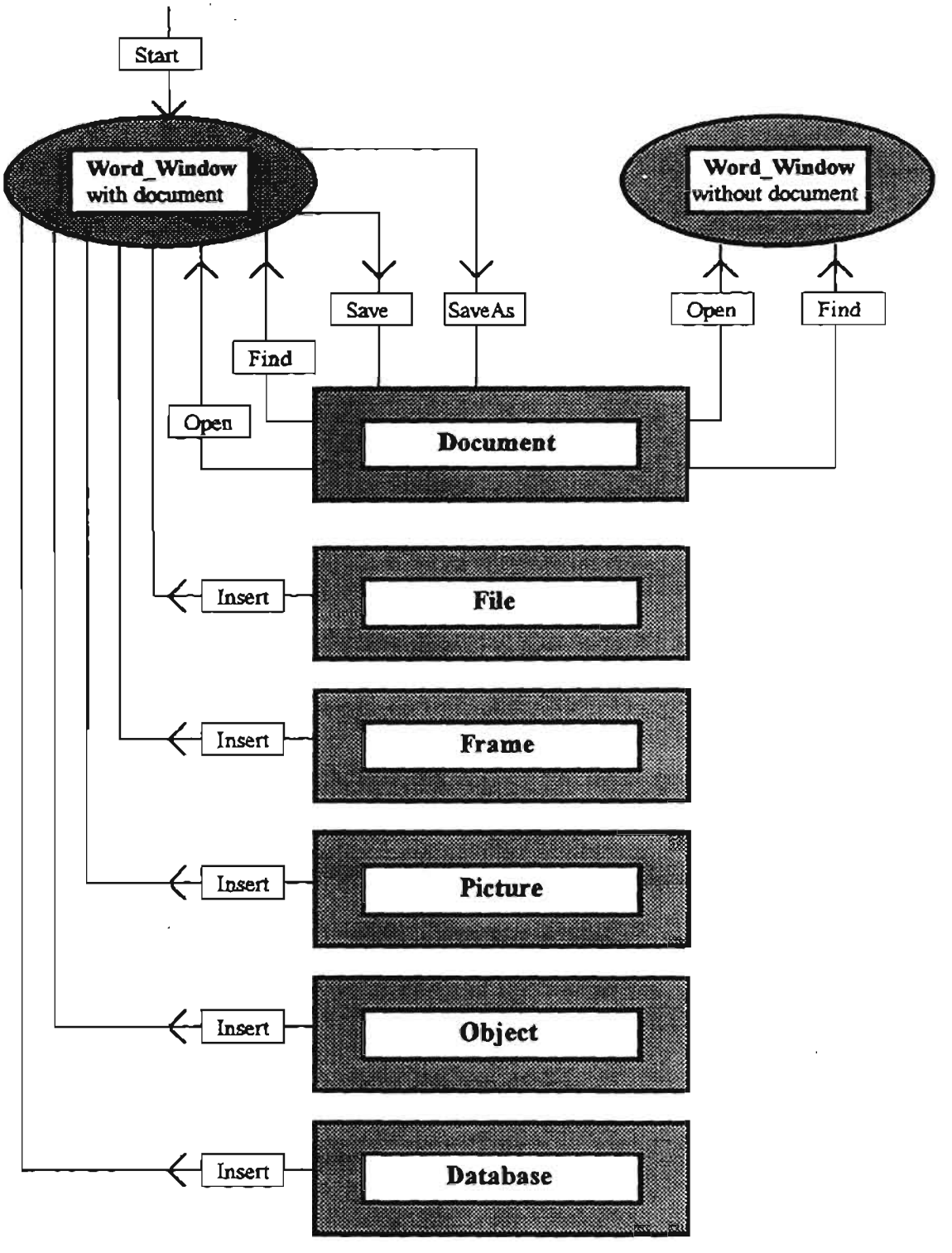



Fig 2. Data base test model fragment for MS WORD

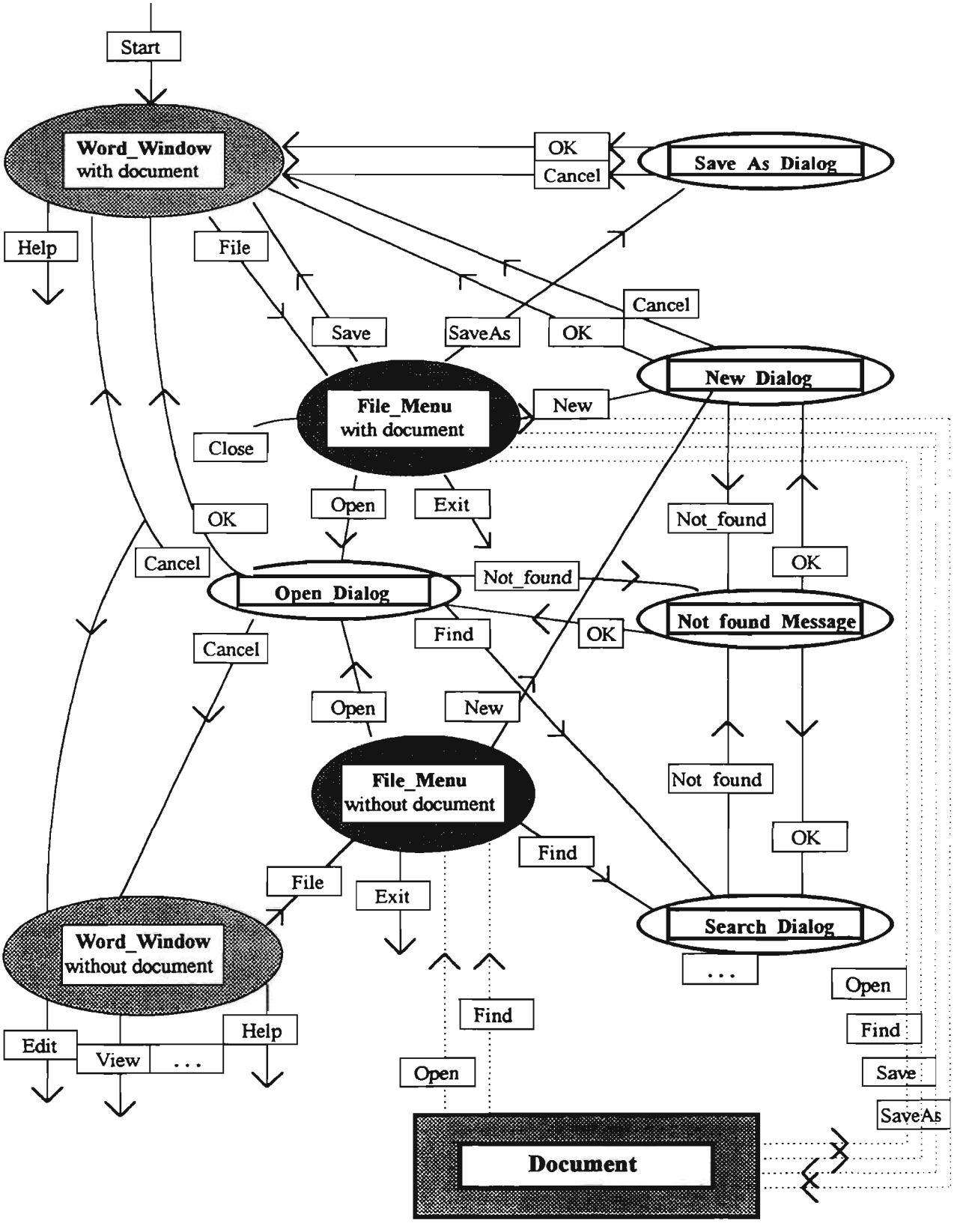



Fig 3. User interface and data base test model fragment for MS WORD

## 5. Test criteria

System coverage models are constructed from internal format of specification. These models are to be covered in prototype execution process:

- CF (Control Flow) criterion. The control graph corresponding to CF criterion made by the TW contains only the essence from point of view of control statements - START, STOP, CASE, IF, WHILE and similar ones - as vertices. The edges in CF graph correspond to linear segments in the program between every pair of the mentioned statements. During prototype execution the control graph is covered, i.e., executed edges are marked so that the user can check how many times any edge is executed.

- UI (User Interface) criterion allows to ascertain how the given tests cover the testable system by means of report forms (RF) and screen forms (SF). Although functionality of the information system is checkable mainly through a system interface, this criterion as well as CF can check correctness of system. The vertices of this graph are procedures and SF or RF called by these procedures. Edges between them specify either procedure calls, or usage of SF or RF. The usage mode (INPUT, MODIFY etc.) is associated with edges. Covering graph has also execution counters associated with edges.

- The idea behind the DB (Data Base) criterion is similar to UI. DB graph shows whether the given tests cover all possible uses of data bases in all possible modes (READ, WRITE, READ-WRITE). The vertices represent procedures and data bases; edges correspond to data base usage. Covering graph has execution counters and usage modes associated with edges.

- DU (Data base and User interface) is a combined criterion - union of UI and DB.

The mentioned criteria are introduced by the authors. However, due to their naturality similar criteria could be independently proposed elsewhere.

## 6. Collection of test suites, test execution and test evaluation

Collection of test suites is carried out the following way. Procedures (executable specifications) selected for testing are executed step by step. The execution can be suspended at user selected points for:

- entering data which will be stored as test suites,
- checking the obtained results and collecting them as check values for the future retesting.

To verify procedure functioning by previously collected tests those procedure statements that provide data input operations can receive data not only from user or data base but also from the stored test. The user, of course, can choose the source of input data: test or the user him/herself. At the user defined checkpoint in the procedure (if it was inserted during test creation), one can

compare the actual obtained results with the check values that were set previously. The result of comparison - exact or partial matching of check values and the actual results - is passed to the user to make a decision about procedure correctness. If modification in procedure seems not vital and the relevant test must be changed, then the possibility to correct the old stored values is offered. If procedure is modified essentially, some difficulties may arise to check procedure by using the old test. In such a case the user has to return to the test creation/collection mode.

The evaluation of test results comprises:

- covering of system coverage models according to the defined testing criteria,
- analysis of system parts not verified by the given test suites,
- collecting of testing statistics - volume of tests and detected differences between actual execution results and check values.

## 7. Possible application to computer program testing

The developed TW is not a new phenomenon in the testing theory. The essence of the TW is its practical aspect, because many testing problems can be resolved as explained above. It provides collection of comprehensive tests in a convenient way for the system design, and definition of expected results of the system operation. Another feature is repeated checking of system correctness with minimal efforts, just using collected tests. One more feature is evaluation of the system testedness degree according to the user selected criteria. A functionally complete test system can be made from collected specification tests.

The ideas of specification testing implemented in the TW could be used also in testing of computer programs, e.g., C++ programs. The collection of test suites and repeated their execution are already implemented in many testing tools. We would like to discuss the possibility to construct various coverage models, different from the traditional ones.

Therefore UI coverage model would have windows of application as vertices and possible actions (operations) with them as edges between vertices. In many cases the UI model can be made automatically in the process of C++ code analysing, especially if standard class libraries are used. There are some situations when automated generating is not feasible and models must be created by hand.

DB coverage model can be used to demonstrate the using of different files through execution of C++ code but, in our opinion, it is not so important and usable. It would be more interesting to show the communication with data storage through SQL queries in such programming languages as INFORMIX, ORACLE etc. In such a case the DB model would consist of program modules as vertices of and data manipulation statements as edges between them.

The coverage procedure of described models would be implemented using some of the popular testing tools. E.g., the Microsoft Test provides a collection of test suites storing it in BASIC-like form thus allowing to analyse them and to cover testing models. Some technical problem could arise in analysing test scripts and establishing precise relations between items of coverage model (mainly vertices) and internal items of test scripts produced by test tool.

Though the approach described in the last part of the paper comprises just ideas we are sure they can be implemented in existing test systems without serious intervention.

**Acknowledgements**

The authors acknowledge many-sided help of Dr.E.F.Miller and Software Research staff that made this paper possible.

**Bibliography**

1. Liggesmeyer P. Modultest und Modulverification: state of the art._Mannheim; Wein; Zurich: BI-Wiss. Verl.,1990. _315 p.

2. Bicevskis J., Borzovs J., Straujums U., Zarins A., Miller E.F.,jr. SMOTL - A System to Construct Samples for Data Processing Program Debugging._IEEE Trans. Software Eng., 1979, SE-5, No.1, pp.60-66.

3. Held G. (ed.) Sprachbeschreibung GRAPES: Syntax, Semantik und Grammatik von GRAPES-86._Verlag: Siemens Aktiengesellschaft, Berlin und Muenchen, 1990._304p.

4. GRADE V1.0 (Windows): Modeling and Development Environment for GRAPES-86 and GRAPES/4GL. User Guide._Siemens Nixdorf Informationssysteme AG, 1993._282p.

# К ВОПРОСУ О ФОРМАЛЬНОМ ОПИСАНИИ ФУНКЦИИ, РЕАЛИЗУЕМОЙ ПРОГРАММОЙ

Я.Я.Бичевский, Ю.В.Борзов

(СССР)

Рассмотрен метод, позволяющий посредством формул наглядно описать работу программ обработки данных.

В последние годы значительное внимание было уделено разработке методов наглядного описания функции, задаваемой программой. Во многих случаях, используя символическую интерпретацию /I,2,3/, удается получить в качестве результата выражение, изображающее ту математическую функцию, которую вычисляет программа. Однако все эти результаты получены для программ вычислительного характера, написанных, как правило, на ФОРТРАНе.

В случае обработки данных (КОБОЛ, ПЛ/I) аргументами и результатами работы программы являются наборы данных; функция, реализуемая программой, представляет собой преобразование одних наборов данных в другие. В докладе предлагается эти преобразования описывать так называемыми формулами обработки файлов (ФОФ). Эти формулы оперируют только такими понятиями, как запись набора данных, часть этой записи (реквизит), отношение порядка между значениями реквизитов, но не используют одно из основных понятий программирования – понятие переменной.

ФОФ описывают работу программы аналогично тому, как регулярные выражения описывают множество слов, акцептируемых конечным автоматом. ФОФ показывают с точностью до числа прохождения циклов все существенно различные режимы работы про-

граммы.

Теоретическими исследованиями доказано, что для программ обработки данных, написанных на языках аналогичных /I/, справедливо следующее:

а) существует алгоритм А, который для любой программы П строит А(П), являющееся ФОФ;

б) если $A(П_1)$ и $A(П_2)$ совпадают, то $П_1$ и $П_2$ функционально эквивалентны.

В докладе показано, что ФОФ можно использовать для наглядного предоставления структуры результирующих наборов данных.

## Литература


I. Я.М.Барздинь, Л.Я.Бичевский, А.А.Калниньш. Построение полной системы примеров для проверки корректности программ, Ученые записки Латвийского государственного университета им.П.Стучки, том 210, Теория алгоритмов и программ, вып.I, стр.I52-I87, Рига, I974.

2. J.C.King. A new approach to program testing, Proceedings 1975 International Conference on Reliable Software, April 1975.

3. W.E.Howden. Symbolic testing and the DISSECT symbolic evaluation system, Computer Science Technical Report No.11 Applied Physics and Information Science University of California, San Diego, May 1976.

АКАДЕМИЯ НАУК СССР
СОВЕТ МОЛОДЫХ УЧЕНЫХ И СПЕЦИАЛИСТОВ ГОРОДА МОСКВЫ

ПЕРВАЯ МЕЖДУНАРОДНАЯ
КОНФЕРЕНЦИЯ МОЛОДЫХ УЧЕНЫХ

# «ПРОБЛЕМЫ ПРОЕКТИРОВАНИЯ И ПРИМЕНЕНИЯ ДИСКРЕТНЫХ СИСТЕМ В УПРАВЛЕНИИ»

Тезисы докладов

Минск 1977

INTERNATIONAL CONFERENCE ON SOFTWARE QUALITY

# ICSQ '95

## Conference Proceedings ICSQ '95



**Editors**

Ivan Rozman
Marjan Pivka

November, 6 - 8 1995

Maribor

Slovenia

# SOFTWARE QUALITY CLASSES: PROBLEM STATEMENT AND THE LATVIAN CASE

**Juris Borzovs**

**Riga Institute of Information Technology**

ABSTRACT:

***ISO, ANSI, IEEE and other standards present variants of quality definitions. Different attributes of quality concept (reliability, portability, learnability, etc.) and measures of them are introduced. Nevertheless, software developers lack comprehensive and recognised, concise practical classification of quality levels. Thus standardised quality level can be neither agreed nor implemented. Draft proposal for practical software quality classification is described.***

***Key words: software quality, quality classes, quality assurance, total quality management.***

## 1. Software Quality Definition and Its Interpretations

(A) The totality of features and characteristics of a software product that bear on its ability to satisfy given needs; for example, conform to specifications. (B) The degree to which software possesses a desired combination of attributes. (C) The degree to which a consumer or user perceives that software meets his or her composite expectations. (D) The composite characteristics of software that determine the degree to which the software in use will meet the expectations of the customer.

The mentioned definition of software quality is one of the most general one[1]. Nevertheless, nobody has found an **universal measure** of quality and will not likely to find it in the future. The reason is very great diversity of systems, products, components, and processes, as well as user needs and expectations. The latter are highly variable, in addition.

According to the international standard [2] quality is characterised by the following six basic attributes: functionality, reliability, usability, efficiency, maintainability and portability.

It is easy to only mention these (or any other) attributes. Much harder - to formulate a precise meaning of an attribute, to choose a measure for attribute evaluation, and to invent a process leading to attainment of desired attribute values.

In the case of software concretization of the mentioned six basic quality attributes collides with several difficulties. Some of attribute values can be found out only at the very end of development process (e.g., reliability and efficiency), when quite often great development resources have already been consumed. Other attributes (e.g., usability) in reality are composed from subattributes (e.g., user interface conformity to GUI requirements, quality of a manual, etc.). Moreover, a well majority of attributes and subattributes can not be expressed quantitatively, but more or less - qualitatively.

Software development process that does not possess quality itself unlikely will result in quality product. Therefore along with a product quality also a process quality is used as an indirect measure of quality of final product [3].

By no means, our view on software quality described so far is neither the only possible nor detailed enough for immediate practical use. However, it is well known that such detailizations exist and are used in practice [e.g., 3].

## 2. A Problem Statement - Need for Predefined Quality Classes

The more attributes, subattributes and measures for quality characterisation we introduce the more thorough our insight in this problem becomes. But at the same time it becomes more fuzzy and more difficult to comprehend. As a result - its practical application also becomes difficult.

At the same time it is quite clear, that more quality product requires greater resources, difference could amount to several times in praxis. A customer does not agree to pay for more resources than he/she absolutely needs. However, the customer usually is not well informed what quality levels could be attained and what is a price of attainment. E.g., flour of the highest class is approximately 30% higher than that of the first class which also could be well used as food. Also there are known minimal requirements and if they are not

satisfied the use of corresponding flour is not secure and is forbidden. A buyer knows well enough flour of what quality he/she should use in specific case and can plan expenses. Software engineering, as far as we know, has not yet come to any specified, well-known and recognised quality classes and specific, measurable parameter values that characterise these classes.

Our goal is to initiate a discussion of this problem and possibly to start some research in order to estimate whether the problem is significant elsewhere, as well as to propose a rough draft of practical solution that is currently exercised in the greatest Latvian software vendor (over 300 employees) *SWH Informatīvās sistēmas* in scope of ISO-9000 certification program.

## 3. Draft Proposal for Quality Classes

We have come to the opinion that a practical quality system should employ usually 5 possible values (or classes of values) of quality features. In case of expert evaluation these would be: 0 - fatal deficiencies or evaluation has not been done at all, 1 - severe deficiencies, 2 - minor deficiencies, 3 - insignificant deficiencies, 4 - without deficiencies. In other cases, where possible, correspondence to specific class (one of the five) will be characterised by specific quantitative or qualitative requirements.

It could be very nice if values of quality features are chosen such that integral (overall) quality could be estimate as follows: if none of feature values of given product is less than N then also the integral quality is not less than N. However, such a system of values has not yet developed. Nevertheless, we are sure that such or similar system of quality classes is needed for systematic statement of quality requirements and for such project management that warrants development of a product belonging to agreed quality class.

Below we (only partly!) demonstrate fragments of values of quality attributes of different classes (0-4).

### 3.1. Product quality

PRD1. Functionality

PRD2. Reliability

PRD2.1. Directly measurable reliability

PRD2.1.1. Mean Time Between Failure

- 0 - lack of measurements
- 1 - over 4 operation hours
- 2 - over 8 operation hours
- 3 - over 40 operation hours
- 4 - over 160 operation hours

PRD2.1.2. Highest observed recovery time

- 0 - lack of observation
- 1 - over 24 hours
- 2 - up to 1 work day
- 3 - up to 1 hour
- 4 - up to 10 minutes

PRD2.2. Indirectly measurable reliability

PRD2.2.1. Testiness

- 0 - there was no documented testing
- 1 - documented complete functional test
- 2 - plus documented complete integration test
- 3 - plus documented complete module C1-test
- 4 - plus documented complete error test

PRD2.2.2. Verifiedness

- 0 - there was no documented verification
- 1 - requirements specification review done
- 2 - plus functional design review done
- 3 - plus functional audit done
- 4 - plus verification and validation plan fulfilled completely

PRD2.3. Security

- 0 - not estimated
- 1 - security class D
- 2 - security class C1
- 3 - security class C2
- 4 - security class B1

PRD4. Efficiency

PRD4.1. Storage consumption

- 0 - not estimated
- 1 - storage consumption over "standard"
- 2 - almost all "standard" storage occupied, parallel tasks impossible
- 3 - parallel tasks are possible in "standard" storage
- 4 - occupies insignificant storage

PRD4.2. Reactivity and/or throughput

- 0 - not estimated
- 1 - on-line operation impossible
- 2 - on-line operation burdensome
- 3 - delay over psychologically admissible, although on-line operation practically possible
- 4 - practically immediate

PRD6. Portability
PRD6.1. Documentation to support portability
PRD6.2. Documentation quality
PRD6.3. Workforce consumption
0 - not estimated
1 - comparable with development from scratch
2 - substantially (at least 50%) less than development from scratch
3 - needs only code recompilation
4 - needs only reinstallation

### 3.2. Process quality

PRC1. Completeness
PRC1.1. Completeness of process documentation
PRC1.4. Audit authority level
0 - not determined
1 - audit by director of department
2 - plus Control group audit
3 - plus external audit
4 - plus external audit by accredited organisation
PRC1.5. User/customer involvement
0 - not expected
1 - customer approves requirements and design specifications
2 - customer representatives are included in Change Control Board
3 - customer representatives directly take part in design and testing
4 - customer and developer act as partners
PRC1.6. Automation level
0 - not determined
1 - CASE tool is being used in design
2 - plus test tools
3 - plus code generation tools
4 - integrated development tool
PRC1.7. Configuration management
0 - not expected
1 - approved configuration management plan
2 - plus Control Change Board, controlled libraries, defined change processing procedure
3 - plus defined configuration management process, CCB checklists
4 - plus outstanding strictness

### 3.3. Documentation quality

PRD3.1. User documentedness
0 - insufficient documentation
3 - advanced quality software documentation
- Operational Concept Description
- Software Requirements Specification
- Detailed Design Description
- Source code
- Executable code
- User manual
- Installation package

## Summary

There is urgent need to have well established and defined software quality classes in order to control software quality assurance activities according to resources available and quality level required. State-of-the-art in software quality is not adequate for practical evaluation of software mainly because of lack of agreed software quality classes. This paper represents a draft proposal for establishing of such classes. By no means, serious discussion and practical research are needed to obtain a common view on the problem.

## References

[1] IEEE Standard Computer Dictionary, 1991.
[2] ISO/IEC 9126. Information Technology – Software product evaluation – Quality characteristics and guidelines for their use, 1991.
[3] PHB: Process Engineering Handbook for Application Software Manufacture and Project Implementation._ Siemens Nixdorf Informationssysteme AG, Muenchen, 1991.
[4] IEEE P1498, Standard for Information Technology - Software Life-Cycle Processes - Software Development: Acquirer-Supplier Agreement, Draft 3.
[5] Fenton N.E. Software Metrics: A Rigorous Approach._ Chapman&Hall, 1991.

# VOL. 2

# PROCEEDINGS OF THE BALTIC WORKSHOP ON NATIONAL INFRASTRUCTURE DATABASES:

## *problems methods experiences*

BALTIC DB

TRAKAI, LITHUANIA
17-20 MAY, 1994


*Editors*

*Janis A.Bubenko Jr.*
*Royal Institute of Technology and SISU*
*Sweden*

*Albertas Čaplinskas*
*Institute of Mathematics and Informatics*
*Lithuania*

*Janis Grundspenkis*
*Riga Technical University*
*Latvia*

*Hele-Mai Haav*
*Institute of Cybernetics of Estonia*
*Academy of Sciences*
*Estonia*

*Arne Sølvberg*
*Norwegian Institute of Technology*
*Norway*



"MOKSLO AIDAI" VILNIUS

# SMOTL—A System to Construct Samples for Data Processing Program Debugging

JĀNIS BIČEVSKIS, JURIS BORZOVS, ULDIS STRAUJUMS, ANDRIS ZARINŠ, AND EDWARD F. MILLER, JR.

*Abstract*—The possibility of automatic construction of a complete set of program tests is considered. A test set system is said to be complete if every feasible program branch (segment) is executed by it. The complete test set construction algorithm for commercially oriented data processing programs is outlined, and the results of its functioning on real programs are analyzed.

*Index Terms*—Analysis of programs, program testing, program validation, symbolic execution, test data generation.

## I. Introduction

IN RECENT YEARS more and more attention has been paid to program validation. One of the common approaches is to test a program on different sets of input data [1]-[6], [8]-[11].

Although absolutely complete program testing can be achievable by executing it on all possible program inputs, most researchers agree that a natural criterion of program testing completeness is the execution of all program branches (or, in other words, traversing all exits of statements) for a finite number of cases [1]-[4], [8].

A path P (a branch P) is called *feasible* if there exist input data which force the program to execute the path P (the branch P). Further on we shall consider only those tests on which the program terminates normally. These are said to be *permissible.*

A finite test set S for the given program will be called a *complete test set* (CSS), if it consists of permissible tests and every program branch that can be executed is executable on some test in S.

The idea of program debugging by CSS construction arises mainly from applications programming experience. The programmer always tries to select a set of tests such that the program executes all its branches. In this paper, we shall discuss only the possibility of automatic CSS construction without any substantiation of this principle.

The SMOTL system constructs a CSS automatically. It differs from other similar systems in two features:

1) SMOTL is oriented to commercially practical programs, whereas other known systems are meant for scientific programs.

2) SMOTL is supplied with a strategy for choosing the program paths to analyze, whereas, in the majority of other systems, the task of selecting such paths is done manually. SMOTL constructs a CSS according to the program text without human interference.

### A. System Survey

The SMOTL system is intended primarily for batch processing of programs written in SMOD; SMOD is a Cobol-like language, but has no means of direct access to data on the external storage devices.

Besides compiling and linkage-editing the SMOTL system contains facilities of the following kinds:

1) detection of certain run-time errors;
2) constructing tests for program validation;
3) to put out samples on the external storage devices;
4) to execute the program repeatedly on the constructed tests (regression testing);
5) to print and compare program output data.

SMOTL is organized logically into six phases, described next.

1) Replacement of the source program by its internal representation in the form of a directed graph.

2) Static analysis of the program with the objective of time and memory space optimization by subsequent SMOTL phases.

3) Construction of *a covering set of paths*, i.e., the test set containing only feasible paths and covering all feasible branches.

4) Minimization of the covering set.

5) Construction of test data for paths from a minimized covering set.

6) Output of test data to an external storage device, compiling and linkage editing of the SMOD test, repeated execution, and printing of generated results. This latter phase has nothing to do with CSS construction and it will not be discussed in the paper.

### B. Problem Solvability

We shall demonstrate in the following that for every real programming language the problem of CSS construction is solvable

Manuscript received June 29, 1977; revised April 7, 1978. The contributions made by the last-named author have been purposely limited in scope to retain as much as possible the original flavor of this work. The objective was to clarify the presentation as much as possible, to uniformize the terminology to that most likely to be understood by readers in the United States, and to clean up a few minor technical details. The unique attitude and approach to problem-solving exhibited by the original manuscript has been retained as much as possible.

J. Bičevskis, J. Borzovs, U. Straujums, and A. Zarinš are with the Department of Computer Science, Latvian State University, Riga, Latvia.

E. F. Miller, Jr. is with Software Research Associates, San Francisco, CA.

0098-5589/79/0100-0060$00.75 

from the theoretical point of view. The crucial point to observe is that every program variable can be assigned only a finite number of different values. To construct a CSS we use an algorithm (described next) for exhaustive search of individual variables' values.

We begin the description of the algorithm by indicating how to check the execution feasibility of an individual path. Subsequently, we shall discuss how the infinite set of all different paths can be treated.

We shall consider only initial paths in this section, i.e., paths starting from the first statement. We execute statements of the path beginning from the first one. If the values of operands are known before the execution of a statement then the execution of this statement is carried out in the usual way by the simulation of program operation.

The values of operands may not be known in two cases:

*Case 1:* When a data input statement is to be executed. In this area the corresponding values are to be assigned from input data, which certainly are not known. We evaluate these variables by assigning some concrete fixed values from the corresponding domains of possible values of the variables. These domains are uniquely determined by the specifications of the variables and this statement. It is important to note that these domains are finite because only numbers of limited length and precision, including strings of limited length, can be represented in the memory of a computer.

*Case 2:* When there is a statement to be executed which does not read values from input data and nevertheless the values of some operands are not known. In this case the path is obviously infeasible and the further analysis is not needed.

The described process terminates in two cases:

*Case A:* When we have succeeded in executing a path from the beginning to the end. Consequently, there will be fixed values for all the input data. This fixation of the values determines the test data which force the program to execute this path. Fixations of this kind are called *feasible fixations.* As soon as we have a sample which forces execution of a path, this path (obviously) is feasible.

*Case B:* When we have not succeeded in the path execution because of the fact that the selected fixation of input data values does not satisfy some conditional statement of the path. In this case, we must repeat execution of the path fixing another value for input data. As soon as the domains of possible values of all variables are finite and every path contains only a finite number of data input statements, the number of all different possible fixations then also becomes finite. The feasibility of the path can be checked by effectively examining all of these fixations.

We have proved that the feasibility of any individual program path can be determined. Now we shall demonstrate how the infinite set of all different initial paths can be treated. We can examine the paths in ascending order of their length and try to select a set of feasible paths having the program exit statement for the last statement of the path (e.g., a STOP path) which covers every program branch. CSS construction will be completed if we succeed in finding such a set.

However, if there is a program branch which has no permissible samples for execution of it then the described procedure does not terminate. This happens by having to consider paths of continually increasing length and waiting till some feasible STOP-path contains the chosen branch. Thus we are forced to analyze an infinite number of paths (except the case of trivial programs without loops).

This apparent difficulty can be avoided, nevertheless. It is clear that at every instant the program's further behavior is uniquely determined by the values of program variables and does not depend on the path on which these values are assigned. More precisely: let feasible paths PA and PB end with the same statement and let the values of the variables at the end of the path PA for a feasible fixation A coincide with the values of the corresponding variables at the end of the path PB for a feasible fixation B. Then, if the path obtained by concatenation of path PA and its continuation PC is feasible, so is also the path obtained by concatenation of paths PB and PC. Note that if two paths differ only in the number of loop traversals and if the value of each variable at the loop exit statement is equal for both paths then there is no need to consider the longer path.

Hence, let us consider only paths with different values of variables at the exit statements of loops. The number of such paths is of course finite; this fact permits us to construct a CSS in all cases.

From the practical point of view, however, the algorithm described cannot be applied directly because of the possible need of an unending exhaustive search. To find more suitable methods for CSS construction we have to take into account that the domains of variables' values are practically infinite. In theoretical investigations [1], [2], [4], [10], [11] the infinity of the domain is assumed explicitly. Under this assumption, the CSS construction problem is theoretically unsolvable. However, the CSS construction algorithm discussed here applies to a wide class of practical data-processing-oriented programs.

The algorithm implemented in SMOTL is similar to the one previously described, but instead of concrete values of variables a generalized description of possible values called a concept "state," is used. It is possible by using some standard optimization methods to reduce the number of different states to a small and manageable number for the case of real-world data-processing-oriented programs. This technique allows us to construct an acceptable CSS in realistic computation times.

## II. Detailed SMOTL Description

### A. Phase I

The first phase of SMOTL operation replaces the source program by a directed graph. The nodes in the graph are the statements of the program and the edges represent the program flow. Every node in the graph contains not only the statement code but also references to the description of operands. Such a representation allows the SMOTL system to traverse the program paths easily and store the program in the core compactly. (About 4K bytes are needed for a program containing 300 statements, to cite a specific example.)

In discussing the next phases of SMOTL operation we shall use the PL/1-style program EXAMPLE to demonstrate the sys-

```
EXAMPLE: PROC OPTIONS (MAIN);
      DECLARE PAY FILE INPUT;
      DECLARE NAME FILE INPUT;
      DECLARE PAYBILL FILE OUTPUT;
      DECLARE 1 P,
                2 CODEP PICTURE '(4)9',
                2 INCOME PICTURE '(4)9V99',
                2 CREDIT PICTURE '(4)9V99';
      DECLARE 1 N,
                2 CODEN PICTURE '(4)9',
                2 NAMEN PICTURE '(4)9V99';
      DECLARE 1 PRINTLINE,
                2 CODE PICTURE '(4)9',
                2 FILL1 CHARACTER (10) INITIAL (' '),
                2 NAMEP CHARACTER (12),
                2 FILL2 CHARACTER (20) INITIAL (' '),
                2 OUTCOME PICTURE '(4)9V99';
L0: OPEN FILE (PAY), FILE (NAME), FILE (PAYBILL);
      ON ENDFILE (PAY) GOTO L9;
      ON ENDFILE (NAME) GOTO L6;
L1: READ FILE (PAY) INTO (P);
L2: READ FILE (NAME) INTO (N);
L3: IF CODEP < CODEN THEN DO; DISPLAY ('NAME NOT FOUND'); GOTO L8; END;
L4: IF CODEP > CODEN THEN GOTO L2;
L5: OUTCOME = INCOME - CREDIT; CODE = CODEP; NAMEP = NAMEN;
      WRITE FILE (PAYBILL) FROM (PRINTLINE); GOTO L8;
L6: CODEN = +9999; GOTO L3;
L7: CODEP = +9999;
L8: READ FILE (PAY) INTO (P); GOTO L3;
L9: CLOSE FILE (PAY), FILE (NAME), FILE (PAYBILL);
END EXAMPLE;
```

Fig. 1. Text of the program EXAMPLE.

tem functioning. This program forms the file PAYBILL by processing the ordered files PAY and NAME. (See Fig. 1.)

### *B. Phase II*

The reader may easily pass over the following definitions of essentially located statements (ELS's) and essential variables without influencing his understanding. One may consider all program statements as ELS's and all program variables as essential variables. For the sake of memory efficiency, however, we have tried to minimize the number of ELS's and essential variables.

The second phase of SMOTL operation selects a set of program statements such that every loop contains at least one statement from this set. We shall call the statement from this set *essentially located statements* (ELS's). Usually the ELS is a loop exit statement. In addition, the first program statement and every program EXIT statement are considered to be ELS's. If several loops have a common part, then the ELS's are selected from this common part in order to minimize the number of ELS. In the program EXAMPLE the statements labeled L0, L3, and L9 are the ELS's.

Associated with every ELS there is a list of variables we call *essential variables associated with the ELS.* A variable x is said to be an essential variable for a certain ELS if there exists a path beginning with this ELS such that the following statements hold true:

1) The variable x is analyzed (referenced) in some conditional statement of this path, or the variable x enters some expression which is analyzed in a conditional statement of this path.

2) The value that the variable x has immediately before execution of the ELS is not changed until it is analyzed in a the conditional statement of the path.

For instance, in the program EXAMPLE the statement L0 has no essential variables because on every path which follows L0 every variable is assigned a new value from the input file before it is analyzed in a conditional statement. L9 has no essential variables because there are no paths following this statement ~~that~~ but conditional statement L3 has the essential variables CODEP and CODEN assigned to it.

Unreachable program statements can be detected along with selection of the ELS's; references to uninitialized variables can be found when identifying essential variables. EXAMPLE contains an unreachable statement L7 but it does not contain references to uninitialized variables.

Finally, the second phase of SMOTL produces a list of all the possible paths from one ELS to another (further referenced as subpaths). In EXAMPLE such subpaths are:

(L0,L1,L2,L3)
(L0,L1,L2,L6,L3)
(L3,L8,L3)
(L3,L8,L9)
(L3,L4,L2,L3)
(L3,L4,L2,L6,L3)
(L3,L4,L5,L8,L3)
(L3,L4,L5,L8,L9)
(L0,L1,L9)

### *C. Phase III*

The third phase of SMOTL tries to construct a set of feasible paths ending with the program exit statement such that this set contains every feasible branch. The most important remaining problem with the method used in SMOTL is to choose a criterion for procedure termination in the case when some program branch is infeasible.

This problem can be solved by the aid of a concept of the *state*—a generalized description of all possible values of a variable. Because of the fact that the construction of a state is highly complicated and depends on the features of concrete programming language we shall only illustrate the construction of a state by simple examples in this paper. States will be used for two purposes.

1) If there are two paths which differ only in the number of loop traversals we shall not consider the longer path in the case when the states associated with the last statements of the paths coincide.

2) When we investigate the feasibility of a path which is a concatenation of paths PA and PB, the concept of the state is used for the possibility of separate analysis of PA and PB. A state contains all the information that must be known in addition to the fact of feasibility of the path PA in order to recognize the feasibility of the path PB. The feasibility of the path PB can be recognized from the fact of feasibility of the path PA and from the set of all states ascribed to the last

statement of PA. Moreover, it is guaranteed that if the paths PA and PB are feasible so is the path obtained by concatenation of PA and PB.

The third phase of the system consists of STRATEGY and ANALYZER. STRATEGY chooses a subpath and passes it on to ANALYZER together with description of relationships of variables at the beginning of the subpath (i.e., its state). ANALYZER's task is to decide if the given subpath is feasible with the given state. If the subpath is feasible, ANALYZER constructs a new state and STRATEGY ascribes it to the last statement of this subpath. Further on STRATEGY selects a new subpath and state and turns to ANALYZER again, etc. The end of the work of the third phase is determined by STRATEGY.

*1)* STRATEGY: STRATEGY begins its operation from the first program statement and the empty state. It selects a subpath to the adjacent ELS and turns to ANALYZER with the question: Is this subpath feasible with the empty state? If the subpath is feasible, then ANALYZER constructs a state which describes the values of essential variables after the execution of the analyzed subpath. Then STRATEGY ascribes the new state to the ELS which is the last statement of the analyzed subpath. Further, STRATEGY selects another subpath which a) follows the subpath investigated before, and b) which has not yet been analyzed with the newly constructed state. ANALYZER investigates the selected subpath with the given state, etc. In that way STRATEGY constructs a feasible path adding one subpath to it on every step. If some subpath turns out to be infeasible then STRATEGY moves one subpath back and selects another subpath to analyze.

The attempted construction of a feasible path in this manner terminates for one of two reasons:

1) the program exit statement is reached;

2) STRATEGY comes to an ELS traversed earlier and the newly constructed state coincides with one of those ascribed to this ELS before and, in addition, every subpath following the ELS has already been investigated (with the same state) in some earlier step.

In the first case STRATEGY includes the constructed path in the covering set of paths. In the second case STRATEGY keeps the constructed path in mind. When construction of some other feasible path is completed STRATEGY examines whether it is possible to add some part of path ending with program exit statement to the prior path. If STRATEGY succeeds the feasible STOP-path obtained will be included into the covering set.

After the construction of one path STRATEGY starts to construct a new feasible path. The starting point of the construction of the next path is an ELS on some path constructed before and chosen so that the number of not-investigated subpath/state pairs is the greatest possible. Here we make use of the second property of states, which allows the process to start the investigation of every path not necessarily from the first program statement, and which analyzes every subpath with every state only once.

*2)* ANALYZER: ANALYZER's task is to decide if the given subpath is feasible with respect to the given state. ANALYZER performs a symbolic execution of a subpath. When a subpath is symbolically executed no values are assigned to the variables, as in the case of normal execution; however, the operations are performed on symbolic values. For instance, the READ statement usually assigns a value from the data set to a variable; in symbolic execution a variable receives a certain symbol for its value. This symbol denotes the location of the corresponding value in the data set. Thus after the path (L0,L1,L2,L3) in the program EXAMPLE the variables CODEP and CODEN will receive the symbolic values $(\text{PAY}, 1\text{-}4)_1$ and $(\text{NAME}, 1\text{-}4)_1$, respectively. $(\text{PAY}, 1\text{-}4)_1$ represents the value located in the first four bytes of the first record in the file PAY; $(\text{NAME}, 1\text{-}4)_1$ represents the value located in the first four bytes of the first record in the file NAME. Symbolic execution of a path results in a system of inequalities which represents the constraints over symbolic values. By checking the consistency of the constraints (solving the system of inequalities) ANALYZER determines the path feasibility. In the given example the subpath (L0,L1,L2,L3) does not contain any conditional statement and the system of inequalities is empty. It has a solution; therefore, the subpath (L0,L1,L2,L3) is feasible.

ANALYZER then constructs the state at the end of the subpath (L0,L1,L2,L3). A state is obtained by direct simplification of the inequality system. (Equalities expressing the values of essential variables at the end of the path are added automatically.) In our example, the program state after the subpath (L0,L1,L2,L3) is the following:

$$\text{CODEP} = (\text{PAY},1\text{-}4)_1$$

$$\text{CODEN} = (\text{NAME},1\text{-}4)_1 .$$

On the next step STRATEGY turns to ANALYZER with the subpath (L3,L8,L3) and with the state constructed before. ANALYZER constructs the system of inequalities

$$(\text{PAY},1\text{-}4)_1 < (\text{NAME},1\text{-}4)_1 ,$$

which obviously has a solution. The newly constructed state is the following:

$$\text{CODEP} = (\text{PAY},1\text{-}4)_2$$

$$\text{CODEN} = (\text{NAME}, 1\text{-}4)_1 .$$

The Only ~~one~~ inequality is ~~in~~cluded because there are no essential variables having $(\text{PAY},1\text{-}4)_1$ for their value. Hence $(\text{PAY},1\text{-}4)_1$ cannot get new logical constraints and although new constraints on $(\text{NAME},1\text{-}4)_1$ may appear the inequality $(\text{PAY},1\text{-}4)_1 < (\text{NAME},1\text{-}4)_1$ always will have a solution because of the practically unrestricted possibility to select the value for $(\text{PAY},1\text{-}4)_1$.

The nonsimplified system of inequalities could be used as a state but we would have too large a number of different states for a program having loops in this case. The main property of the simplified system of inequalities is the following: if both the given and the simplified systems (obtained for path PA) have some inequalities (obtained for an arbitrary continuation of PA) then they have solutions simultaneously, but the individual solutions may differ.

The further behavior of the program depends only on relationships between the values of the variables. It does not depend on concrete values of the variables (in our case con-

crete symbolic values). Therefore, we can introduce the following definition of coincidence of states.

Two states are said to coincide if the corresponding systems of equalities and inequalities are equal up to names of the symbolic values.

Obviously, in our example, the newly constructed state coincides with the one constructed before. Therefore, STRATEGY selects the subpath (L3,L8,L9) instead of the subpath (L3,L8,L3) for further investigation. This subpath will appear to be feasible; thus STRATEGY will finish the construction of the first feasible STOP-path (L0,L1,L2,L3;L8,L3;L8,L9).

In general, the construction of a state merely consists of simplification of the system of inequalities obtained by the symbolic execution of a path.

One of the permitted simplifications has already been demonstrated in the example. There are many analogous simplifications and their formal description is given in [1], [4], [11]. Because of the high complexity of these descriptions we do not discuss this matter further in this paper.

In some cases, the system of inequalities built by ANALYZER may turn out to be nonlinear, in which case finding a solution is an unsolvable problem. Practical experience suggests that such nonlinear systems seldom appear in data processing. A method to solve systems of inequalities was developed for the SMOTL system. This method does not guarantee the finding of a solution in all the cases even when it is known to exist. However, the method is very fast; the essence of the method is described next.

First, assume that every variable from the inequalities has been ascribed a segment of its possible values. This segment is determined by the specification of the variable and by the inequalities, in which the variable is compared with constants. For instance, if a variable is specified by the statement

DECLARE X DECIMAL FIXED (2)

and the system of inequalities contains the inequality X>13, then the variable X is contained in the segment [14, 99].

Recall that the system of inequalities is being repeatedly examined; for every inequality and for every variable in the inequality the segment corresponding to this variable is "corrected." For instance, if the inequality is x<y, the corresponding segments for variables x and y are [a,b] and [c,d], and x and y are declared to be integers, then after the correction the segments are [a, min (b,d - 1)] and [max (a+1,c),d].

The procedure terminates when the current pass examining the inequalities does not change any segment or when some segment becomes empty. In the latter case, the system of inequalities has no solution and, therefore, the analysis of it is finished. In the former case, if the system of inequalities contains no arithmetical expressions then the variables will have been assigned the lower bounds of the corresponding segments. The assigned values represent the solution of the system.

However, if the system of inequalities contains arithmetic expressions, then the solution is found by a restricted search of the values from the segments (the exhaustive search may take too much time). If ANALYZER does not succeed in this restricted search then it asserts (perhaps incorrectly) that the given subpath is infeasible. Consequently, it cannot be guaranteed that the sample system constructed by SMOTL is complete in these cases.

*3) Additional Remarks:* Every path which is recognized by STRATEGY to be feasible and which eventually reaches the program exit statement is added to the covering set of path being constructed. When added, a check is made to determine whether every program branch is already represented in the set. If it is, the third phase terminates. However, if the program contains an infeasible branch, STRATEGY is forced to check all the possibilities, i.e., every subpath with every state.

The third phase terminates in two more cases: 1) lack of memory for storing of the newly constructed state (SMOTL can store approximately 200 states on the average); 2) the time allowed for the work of the third phase is spent (typically, 5 min of CPU time is allowed for the third phase of SMOTL). As mentioned earlier, in these two cases it is not guaranteed that the constructed test set is complete.

Analyzing the program EXAMPLE, STRATEGY produces the following set of feasible paths:

(L0,L1,L2,L3;L8,L3;L8,L9)
(L0,L1,L2,L3;L4,L2,L3;L8,L9)
(L0,L1,L2,L3;L4,L2,L6,L3;L8,L3;L8,L9)
(L0,L1,L2,L3;L4,L5,L8,L3;L8,L9)
(L0,L1,L2,L3;L4,L5,L8,L9)
(L0,L1,L2,L6,L3;L4,L5,L8,L3;L8,L9)
(L0,L1,L2,L6,L3;L4,L5,L8,L9)
(L0,L1,L9).

Here a semicolon separates the subpaths within each path.

### D. Phase IV

In batch debugging it is not convenient to deal with a large number of tests. Therefore, the fourth phase rearranges and rejects certain paths produced by the third phase. The aim is to reduce the number of paths in the covering set.

For instance, the paths (L0,L1,L2,L3;L8,L3;L8,L9) and (L0,L1,L2,L3;L4,L2,L3;L8,L9) can be merged into one path (L0,L1,L2,L3;L4,L2,L3;L8,L9). Here we make use of the fact that the program states ascribed to ELS L3 after traversing the paths (L0,L1,L2,L3), (L0,L1,L2,L3;L8,L3), and (L0,L1,L2,L3;L4,L2,L3) coincide. It is easy to check that the newly constructed path contains the same branches as the two given paths. Due to the properties of states, the feasibility of the newly constructed path is guaranteed.

We shall obtain as a result for the program EXAMPLE a covering set of paths, which contains only these two paths:

(L0,L1,L2,L3;L8,L3;L4,L2,L3;L4,L5,L8,L3;L4,L2,L6,L3;L8,L3;L8,L9)

and

(L0,L1,L9).

### *E. Phase V*

The fifth phase constructs test data values for the paths selected by the preceding phases. The only data obtained at analysis time contains information about a very few values of input records, because a significant portion of them have been rejected when the states were constructed. Therefore, the symbolic execution of the entire path is performed once more to obtain a system of inequalities in all-symbolic-values form. By solving these inequalities we obtain the values for the input which forces the program to execute the corresponding path.

For the program EXAMPLE the complete test set is:

$T_1$

| file PAY | file NAME |
|---|---|
| 1st record 0001111111000000 | 1st record 0002 J. BORZOVS |
| 2nd record 0003222222333333 | 2nd record 0003 J. BICEVSKIS |
| 3rd record 0004444444444444 | |
| 4th record 0005000000000000 | |

$T_2$

| file PAY | file NAME |
|---|---|
| empty | empty |

## III. Implementation Notes

The SMOTL system was implemented during 1974-1976 on a Soviet computer "MINSK-32" (160K bytes main storage, CPU speed approximately 50 000 operations/s) [7]. Direct-access devices were not used. The selection of this environment was the result of our aim to construct a practical test data generation system. The SMOTL system consists of 30 000 computer instructions.

CSS construction time usually does not exceed the program's compile time. We have analyzed the results of the system working on 39 programs. They were selected at random from the software of several already implemented management systems. It turned out that our system processed programs having fewer than 300 statements sufficiently well. The programs contained an average of 11 conditional statements. The CSS was constructed for 16 programs out of 25; for the rest the test data obtained covered about 74 percent of all the feasible paths.

The situation was considerably worse in the case of programs containing more than 300 statements (there were 14 such programs). Practically useful results were obtained only for 5 of them. The average number of conditional statements in programs from this class was 66. Five programs contained from 400 to 500 statements, and 7 had more than 500 statements.

Time and storage resources have proved to be insufficient for the processing of very long programs. Our concept of "state" proved to be insufficient in the same cases, when highly complicated means were employed in a program such as irregular references to array elements or complicated arithmetic expression calculations.

## IV. Conclusions

The advantage of such systems as SMOTL lies in generalized program testing. In testing with the constructed test data, the system demonstrates the function of all program parts that can be executed without abnormal termination. The SMOTL system gives diagnostic messages explaining the reasons of infeasibility of all the remaining (untested) program parts. SMOTL is especially convenient in batch-processing mode, where normal program termination gives a programmer much more information than its abnormal stop.

The current SMOTL implementation shows that for many data-processing style programs it is possible to construct CSS in acceptable time on widely used computers. We hope to obtain similarly positive results in developing our system for PL/1.

## References


[1] J. M. Barzdin, J. J. Bičevskis, and A. A. Kalninsh, "Construction of complete sample system for testing correctness of programs," *Ucenye Zapiski Latv. Gos. Univ.* (Riga, Latvia), vol. 210, pp. 152-188, 1974.

[2] A. A. Kalninsh, J. J. Bičevskis, and J. M. Barzdin, "Solvable and unsolvable cases of the problem of construction of a complete sample system," *Ucenye Zapiski Latv. Gos. Univ.* (Riga, Latvia), vol. 210, pp. 188-206, 1974.

[3] E. F. Miller and M. R. Paige, "Automatic generation of software test-cases," in *Eurocomp Conf. Proc. 1974*, 1974, pp. 1-12.

[4] J. M. Barzdin, J. J. Bičevskis, and A. A. Kalninsh, "Construction of complete sample systems for correctness testing," in *Mathematical Foundations of Computer Science*. Berlin: Springer, 1975, pp. 1-12.

[5] J. S. King, "A new approach to program testing," in *Proc. 1975 Int. Conf. on Reliable Software*, Apr. 1975.

[6] W. E. Howden, "Methodology for the generation of program test data," *IEEE Trans. Comput.*, vol. C-24, pp. 554-559, May 1975.

[7] A. P. Ershov, "A history of computing in the USSR," *Datamation*, vol. 21, pp. 80-88, Sept. 1975.

[8] W. E. Howden, "Symbolic testing and the DISSECT symbolic evaluation system," Computer Science Tech. Rep. 11, Applied Physics and Information Science, Univ. California, San Diego, May 1976.

[9] L. A. Clarke, "A system to generate test data and symbolically

execute programs," *IEEE Trans. Software Eng.*, vol. SE-2, Sept. 1976.
[10] J. M. Barzdin, J. J. Bičevskis, and A. A. Kalninsh, "Automatic construction of complete sample system for program testing," in *1977 IFIP Congr. Proc.*, pp. 57–62, 1977.
[11] J. J. Bičevskis, "Automatic construction of sample systems," *Programmirovanije* (Moscow, USSR), no. 3, pp. 60–70, 1977.

**Jānis Bičevskis** graduated from the Faculty of Physics and Mathematics, Latvian State University, Riga, USSR, in 1970. His first publications dealt with the theory and complexity of computation.

During his studies he designed and implemented a package for automation of commercially oriented program development. Since 1970 he has been with the Department of Software, Computer Center, Latvian State University. He was the Leader of design and implementation of commercially oriented language SMOD and was one of its compiler's architects. (He was presented with the Y.C.L. Award for this work in 1974.) He participated in design and implementation of five data management systems and originated a new fast sorting technique and was the Head of the SMOTL project. Since 1972 he has published more than 20 papers on program testing, especially on symbolic execution and test data generation. He prepared a dissertation for the Candidate of Science degree in physics and mathematics in 1977 and is now the Director of the Production Sector, Computing Center, Latvian State University. His research interests include the theory of program testing and verification, programming languages, programming methodology, sorting methods, and system software.

**Juris Borzovs** graduated from the Faculty of Physics and Mathematics, Latvian State University, Riga, USSR, in 1973. His first publications dealt with the theory of holographical sets.

During his studies he designed and implemented a preprocessor for a data processing language. Since 1973 he has been with the Department of Software, Computing Center, Latvian State University (from 1973 to 1974 as a Mathematician and Programmer, from 1974 to 1975 as a Senior Mathematician and Programmer, and since 1975 as a Senior Research Associate). Since 1973 he has also served as a part-time Lecturer at the Latvian State University. He participated in the design and implementation of several data management systems, a system of scientific measurements evaluation, a telecommunication subsystem, and the SMOTL system. His research interests include the theory of program testing and verification, programming methodology, and system software.

**Uldis Straujums** graduated from the Faculty of Physics and Mathematics, Latvian State University, Riga, USSR, in 1973. His first publications dealt with limiting recursive functions.

During his studies he designed and implemented a preprocessor for a data processing language. Since 1973 he has been with the Department of Software, Computing Center, Latvian State University (from 1973 to 1974 as a Mathematician and Programmer, from 1974 to 1975 as a Senior Mathematician and Programmer, and since 1975 as a Senior Research Associate). Since 1973 he has also served as a part-time Lecturer at the Latvian State University. He participated in the design and implementation of several data management systems, a system of scientific measurements evaluation, a telecommunication subsystem, and the experimental SMOTL system. Now he is Director of the PL/1 interpreter project. His research interests include the theory of program testing and verification, programming languages, and system software.

**Andris Zarinš** graduated from the Faculty of Physics and Mathematics, Latvian State University, Riga, USSR, in 1970. His first publications dealt with the theory of recursive functions.

Since 1970 he has been with the Department of Software, Computer Center, Latvian State University. He serves as a Senior Research Associate. Since 1971 he has also served as a part-time Lecturer at the Latvian State University. He participated in the design and implementation of the commercially oriented language SMOD and has been one of its compiler's architects. (He was presented with the Y.C.L. Award for this work in 1974.) He was also involved in the development of three data management systems, a telecommunication system, and the experimental SMOTL system. He is now participating in the development of the PL/1 interpreter and the design of testing and verification system for the PL/1 language. His research interests are in the theory of program testing, operating systems, programming languages, and system software.

**Edward F. Miller, Jr.** received the B.S.E.E. degree from Iowa State University, Ames, in 1962, the M.S. degree in applied mathematics from the University of Colorado, Boulder, in 1964, and the Ph.D. degree from the University of Maryland, College Park, in 1968.

He was an Instructor at the University of Maryland from 1964 to 1968. He was previously Director of the Software Technology Center, Science Applications Inc., San Francisco, and Director of the Program Validation Project at General Research Corporation, Santa Barbara, CA. He is currently an independent consultant and lecturer and is Technical Director of Software Research Associates, San Francisco, CA, a firm devoted to advanced technology and software applications. His interests include software engineering management, software testing technology, automated tool design and development, hierarchical design and implementation methods, and computer architecture.

Dr. Miller is a member of the IEEE Computer Society, the Association of Computing Machinery, the Society of Industrial and Applied Mathematics, and several honorary societies. He is currently Editor of *Computer Architecture News* (CAN), and Associate Editor of *Computer Magazine*.

Part 3 - Techniques for the Management of IT Security.
[12] ISO/IEC JTC 1/SC 27: PDTR 13335-1 Guidelines for the Management of IT Security (GMITS); Part 1 - Concepts and Models for IT Security.
[13] Järvinen and Kerola P.: Systemointi I. OY Gaudeamus AB, Helsinki, Finland, 1978.
[14] Kaijser, P.: Data protection in Communications and Storage. Proc. IFIP TC11, IFIP/Sec'95.
[15] Karila, A.: Open Systems Security - An Architectural Framework. Doctoral Dissertation. Telecom Finland, Business Systems R&D. Helsinki, Finland, 1991.
[16] LaPadula, L.J.: Rule-Set Modeling of Trusted Computer System in Abrams et al: Information Security - An Integrated Collection of Essays. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.
[17] Leiwo, J.: Deterrence of Computer Network Crime: The International Coordinating Level Approach Towards Legislation. Univ. of Oulu, Dept. of Info. Proc. Science, Working Paper Series B 35. Oulu, Finland, Jan. 1995.
[18] Leiwo, J.: Deterring Computer Network Criminals with Legislative Methods - The Need for International Harmonization. Proc. 2nd Groningen International Information Technology Conference for Students. Univ. of Groningen, Netherlands, 1995.
[19] Olson, I.M. & Abrams, M.D.: Information Security Policy in Abrams et al: Information Security - An Integrated Collection of Essays. IEEE Computer Society Press, Los Alamitos, California, USA, 1995.
[20] Parker, D. B.: A New Framework for Information Security to Avoid Information Anarchy. Proc. IFIP TC11, IFIP/Sec'95.
[21] Parker, D. B.: Managers Guide to Computer Security. Prentice Hall Company. Reston, VA, USA, 1981.
[22] Proposal for Council Recommendation on Common Information Technology Security Evaluation Criteria (ITSEC). Com(92) 298 Final. Office for Official Publications of the European Communities. Brussels, Belgium, 10.9.1992.
[23] Atkinson, R.: Security Architecture for the Internet Protocol. RFC1825, August 1995.
[24] Stallings, W.: Network and Internetwork Security: Principles and Practice. Prentice Hall, Inc. Englewood Cliffs, NJ, USA, 1995.
[25] Stallings, W: SNMP, SNMPv2, and CMIP: the practical guide to network management standards. Addison-Wesly Publishing Company, Inc., USA, 1993
[26] Stevens, R.W.: UNIX Network Programming. Prentice Hall Inc., Englewood Cliffs, NJ, USA, 1990.
[27] Tanenbaum, A.: Modern Operating Systems. Prentice-Hall, Inc. Englewood Cliffs, NJ, USA, 1992.
[28] Trusted Computer System Evaluation Criteria (TCSEC). USA Department of Defense. CSC-STD-001-83, Library NR. S225. August 15, 1983.
[29] Williams, J.G. & Abrams, M.D.: Formal Methods and Models in Abrams et al: Information Security - An Integrated Collection of Essays. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.

# Framework for Potential Latvian Software Engineering Standards

**Anda Adamsone , Juris Borzovs, Rudite Cevere, Marija Luckina**

Riga Information Technology Institute
Skanstes iela 13, LV-1013 Riga, Latvia
phone: +371-2-362400, fax: +371-7821457
e-mail: BORZOVS@SIS.LV

**Abstract**

Every software and/or software quality manager is confronted with questions like the following: what set of standards to choose, what standards implementation begin from, what standards set should be created and/or used to obtain complete software quality infrastructure, etc.

Research done in Riga Information Technology Institute shows that neither ISO nor any other publicly available set of software engineering standards represent a comprehensive standards framework. However, IEEE Std 1498/EIA IS 640 together with ANSI/IEEE Software Engineering Standards Collection and several ISO standards could comprise a rather stable fundament for software engineering standards architecture.

## 1 Introduction

There are isolated software engineering standards addressing separate areas of software system engineering, such as safety, quality, configuration management, software, etc. These standards come from different national, international, and other organisational bodies and are born of local needs. Some areas of a system are not covered by standards. Some standards address inappropriate system areas. In some cases, these standards have been marching with their unique terminologies and procedures. Consequently, there are multiplicities, overlaps, and inconsistencies among these system-level standards. These unique, local, or self-serving standards furnish, at the best, a fragmented approach to software system engineering. Standards that address the total life cycle of a system are not available.

There is a growing realisation among the engineering communities that an embracing system life cycle framework is needed. From economic viewpoints, system managers are reaching similar realisation. However, a forum for addressing these system-level issues and needs is not yet available or not known.

Software engineering standards (SES) obviously is an agreement (as any other standards, of course) between parties: customers, developers, operators, users, managers, etc. It is also a kind of watershed between the user needs and expectations, and the developers practical ability to fulfil them. A set of standards should be considered also as (rather fuzzy) measure of software engineering (SE) or any other sector maturity: more processes, products and other features standardised - more mature the sector.

## 2 Why not Simply Take the ISO Standards?

Available ISO standards do not constitute comprehensive set of SE standards. Although the set contains several tens of standards, most of them are of limited use: vocabularies [1,2], constructs and conventions for representation [3-12]. Some standards are devoted to documentation [13-15], quality issues [16-19], configuration management [20], security [21] and other [22]. We intentionally do not mention here programming language standards (Ada, C, SQL/Ada etc.) because they are not used in everyday programming. It can be realised easily that ISO SE-related standards do not cover significant SE project functions (e.g., requirements specification, design, implementation, verification and validation, etc.) and project documentation, except that of users. Existing standards are greatly dispersed, they also lack any umbrella-standard (something like [17] or the very late [23], but more technological and detailed one).

The conclusion could be that ISO SE-standards alone are still long way from needed condition.

## 3 Who Produces Software Engineering Standards?

Concluded that ISO are not able to provide entire set of SE standards we are to look for other potential standards provider [24,25]. We can observe from Fig. 1, that in spite of a number of SE standardisation bodies there are few principal standards

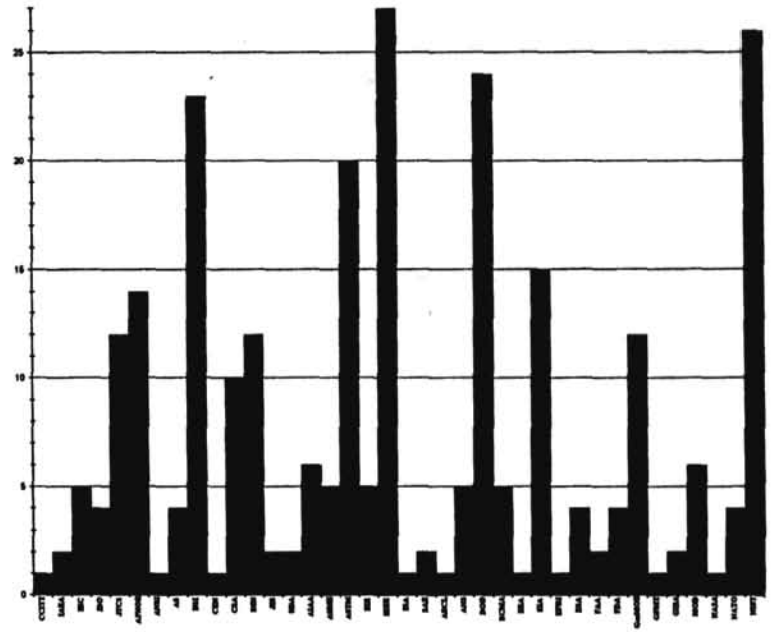


Fig. 1 SES by standardisation body

developers: British Standards Institution (BSI), American Society for Testing and Materials (ASTM), Institute of Electrical and Electronic Engineers (IEEE), US DOD, and National Institute for Standards and Technology (NIST), to name the most productive ones. It should be also stated that over 30 SES projects are in-progress under supervision of Joint Technical Committee 1 of International Organisation for Standardisation (ISO) and International Electrotechnical Commission (IEC). IEEE conducts over 15 projects. However, these projects are at different levels of maturity and sometimes last even for 10 years (3-6 years are quite typical). We also can observe (see Fig.2) that different activities are not uniformly covered by standards. Significant efforts are devoted to development, documentation, quality assurance, but very few - to maintenance, review, audit and problem resolution.

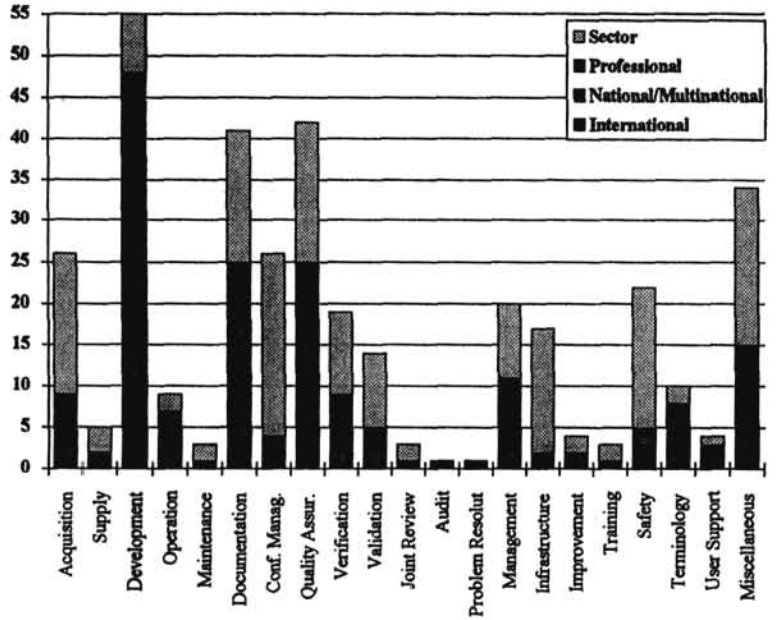



Fig. 2 International, National/ Multinational, Professional and Sectoral SES by SE activities

# 4 Four-level Standards Framework

As soon as we have no ready-to-use comprehensive, well-structured and complete set of standards from ISO or any other known organisation we have to either make such a set from scratch or try to combine existing rather isolated standards into one set. The first alternative can not be viewed as time- and cost-effective one, at least until the second one is tried.

We propose, and it should not be a surprise, to take the ISO 9000-3 as an umbrella-standard (see Fig. 3). Although this standard is the only sector-specific one out of ISO 9000 scope nevertheless its several tens of pages provide too few information on software development process and software products. To overcome such a deficiency the next, more detailed level of SE standards could be represented by ISO 12207 and its more elaborated counterpart - IEEE 1498/EIA IS 640. The latter one is very close to US DoD MIL-STD-498. The first part (approx. 50 pp.) of the standard describes typical

software development activities and requirements. The second part (approx. 150 pp.) describes required contents and formats of needed software products (plans, descriptions, reports etc.). This second part together with previously adopted IEEE SE standards [27-38], the well-known security "orange book" [39] and its potential off-spring [40], related GUI (Graphical User Interface) standards and several ISOs [14-16,18,21] could well comprise the third level of standards framework, leaving the fourth level for corporate (organisation-wide) management, development and support procedures and guides.

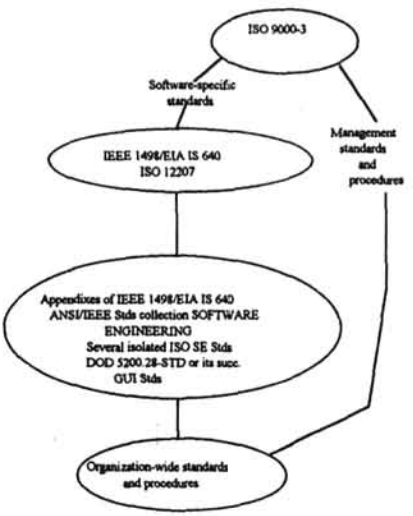



**Fig. 3 SE standards 4-level framework**

# 5 The Future

Latvian National Standardisation and Metrology Centre - a member body for ISO and CEN - has (among others) established the Information Technology Standardisation Technical Committee (IT STK) to act as a counterpart of the ISO/IEC JTC1 (Joint Technical Committee One) and to approve national IT standards. IT STK declared the ex-USSR ESPD standards to be obsolete and began to develop new or adapt existing international or professional IT standards. The most active work is observed towards Latvian-specific (code tables, fonts, keyboard layout, etc.) and SE standards. The first 11

SE standards based on IEEE SE set have been already affirmed as national standards. The framework for future SE standards described briefly in this paper is under consideration at IT STK.

## References


[1] ISO/IEC 2382-7:1989 Information technology - Vocabulary - Part 07: Computer programming.

[2] ISO/IEC 2382-20:1990 Information technology - Vocabulary - Part 20: System development.

[3] ISO 3535:1977 Forms design sheet and layout chart.

[4] ISO 5806:1984 Information processing - Specification of single-hit decision tables.

[5] ISO 5807:1985 Information processing - Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts.

[6] ISO 6593:1985 Information processing - Program flow for processing sequential files in terms of record groups.

[7] ISO/IEC 8211:1994 Information technology - Specification for a data descriptive file for information interchange.

[8] ISO/IEC 8631:1989 Information technology - Program constructs and conventions for their representation.

[9] ISO 8790:1987 Information processing systems - Computer system configuration diagram symbols and conventions.

[10] ISO/IEC 11172-4:1995 Information technology - Coding of moving pictures and associated audio for digital storage media up to about 1.5 Mbit/s - Part 4: Conformance testing.

[11] ISO/IEC 11179-4:1995 Information technology - Specification and standardisation of data elements.

[12] ISO/IEC 11411:1995 Information technology - Representation for human communication of state transition of software.

[13] ISO 6592:1985 Information processing - Guidelines for the documentation of computer-based application systems.

[14] ISO 9127:1988 Information processing systems - User documentation and cover information for consumer software packages.

[15] ISO/IEC TR 9294:1990 Information technology - Guidelines for the management of software documentation.

[16] ISO/IEC 9126:1991 Information technology - Software product evaluation - Quality characteristics and guidelines for their use.

[17] ISO 9000-3 Guidelines for the application of ISO 9001 to the development, supply and maintenance of software.

[18] ISO/IEC 12119:1994 Information technology - Software packages - Quality requirements and testing.

[19] ISO 10013:1995 Guidelines for developing quality manuals.

[20] ISO 10007:1995 Quality management - Guidelines for configuration management.

[21] ISO/IEC 9798-4:1995 Information technology - Security techniques - Entity authentication - Part 4: Conformance testing.

[22] ISO/IEC 14102:1995 Information technology - Guideline for evaluation and selection of CASE tools.

[23] ISO/IEC 12207:1995 Information technology - Software life cycle processes.

[24] Master Plan for Software Engineering Standards. Prepared by Software Engineering Standards Long-Range Planning Study Group. December 1, 1993. Version 1.0. 71p.

[25] Survey of Existing and In-Progress Software Engineering Standards. Prepared by Software Engineering Standards Long-Range Planning Study Group. December 1, 1993. Version 1.0. 51 p.

[26] IEEE Std 1498/EIA IS 640 Trial use standard - Standard for Information technology - Software life cycle processes - Software development - Acquirer-supplier agreement.

[27] ANSI/IEEE Std 1008-1987, IEEE Standard for Software Unit Testing.

[28] ANSI/IEEE Std 1012-1986, IEEE Standard for Software Verification and Validation Plans.

[29] ANSI/IEEE Std 1016-1987, IEEE Recommended Practice for Software Design Descriptions.

[30] ANSI/IEEE Std 1028-1988, IEEE Standard for Software Reviews and Audits.

[31] ANSI/IEEE Std 1042-1987, IEEE Guide to Software Configuration Management.

[32] ANSI/IEEE Std 1058.1-1987, IEEE Standard for Software Project Management Plans.

[33] ANSI/IEEE Std 1063-1987, IEEE Standard for Software User Documentation.

[34] ANSI/IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.

[35] ANSI/IEEE Std 730.1-1989, IEEE Standard for Software Quality Assurance Plans.

[36] ANSI/IEEE Std 828-1990, IEEE Standard for Software Configuration Management Plans.

[37] ANSI/IEEE Std 829-1983, IEEE Standard for Software Test Documentation.

[38] ANSI/IEEE Std 830-1984, IEEE Guide for Software Requirements Specifications.

[39] Trusted Computer Security Evaluation Criteria, Department of Defence, DOD 5200.28-STD (1985)

[40] J.Cugini. The Common Criteria: On the road to international harmonization. Computer Standards & Interfaces 17 (1995), 315-320.

Hele-Mai Haav, Bernhard Thalheim (Eds.)

# Databases and Information Systems

Proceedings of the
Second International Baltic Workshop
Tallinn, June 12-14, 1996

Volume 2: Technology Track

# Recent IT&T Legislation: Trends in Latvia

**Juris Borzovs, Dr. comp. sci.**

**Director, Riga Information Technology Institute (RITI), Latvia**

**Ilgvars Imša**

**Head of the Legal Department, DATI Group, Latvia**

**Uldis Ķinis**

**Chief Judge of Kuldiga Region Court, Latvia**



***There was a remarkable upsurge in legislative activity in Latvia in 1997. The country's declared aim of joining the European Union has provided a number of working groups with very specific tasks in drafting new laws, among them some in the IT&T sphere. Specific examples include proposed amendments to the law on copyright and related rights, as well as a new draft law on the protection of data bases. These have been elaborated in response to a needed to adapt EC directives and various international conventions. The work is occurring under the auspices of the Ministry of Culture. The Ministry of Transport, meanwhile, is working on extensive changes to the law on telecommunications, as well as on a new law on informatics, a new law on personal data protection, and possibly some other legislation, as well. The Ministry of Justice is dealing with amendments to the criminal code in order to provide sanctions against computer-related crime. This paper provides an overview of these activities, the underlying principles of the various draft laws, as well as the extent to which new Latvian legislation is conforming to EC directives and international conventions which Latvia has already joined or is planning to join (the WIPO agreement, TRIPS, etc.).***

## LEGISLATIVE ACTIVITY IN 1997

1997 was a year of purposeful activity in Latvia in terms of dealing with legislation in the field of informatics and in related sectors. For the first time, we saw significant government interest and support for this process, even though the financing which was provided was scant (working groups received USD 5,000 apiece to draft new laws). However, the very fact that three separate ministries (Transport, Culture and Justice) were involved in drafting new laws or amendments to existing ones signified that the ice has been broken in this area. Presumably of considerable importance in the ongoing development of informatics in Latvia will be the national program "Informatics", which was the object of careful work throughout 1997 and which should be approved in 1998. The program contains a section about adjustment of legislation. In fact this process has already begun, without waiting for approval of the program, and that is only natural.

Still, achievements which have been made in the area of legislation so far are far from sufficient, because over the course of the entire year of 1997, Parliament did not adopt a single law in the IT&T field, and that is not because all necessary legislation has already been adopted. Deputies may argue that the Cabinet of Ministers did not submit any proposals concerning informatics legislation, so there was nothing to be approved. That is true, but it does not tell the whole story. The "Latvijai" faction of Parliament submitted legislation "On information, its utilization and protection", in 1996. The proposal had two sections: the first specified that all state and local government information is fundamentally available to everyone and set out the order for access; the second section addressed the issue of government secrets.

The legislation was of adequate quality, but Parliament rejected it, probably because the "Latvijai" faction is controversial and in opposition, and there is a prevailing stereotype that its deputies are not able to offer any sensible proposals. In this case, however, the baby was thrown out with the bathwater, and even though Parliament adopted a law on state secrets at the very end of the year, the principle of openness in the field of government information is not yet enshrined in law.

So what accomplishments were made in 1997? Working groups from the three aforementioned ministries have completed work on a new law on informatics, as well as amendments to the law on copyright and related rights, as well as the criminal code. These drafts are currently being circulated among other ministries for their comments, and if this process ends successfully, the Cabinet of Ministers will be able to submit the drafts to Parliament. The working groups will be able to go on to other work.

## THE TAXONOMY OF INFORMATICS LAWS AND REGULATIONS

It must be said that work in the area of IT&T legislation in Latvia has largely been uncoordinated, and it has been based mostly on the go-vernment's political desire to react to the European Union's various directives. There has been no attempt to provide a conceptual underpinning for this legislative activity, but this must be done in order to ascertain that the new laws are not only in concert with EU directives (this must not be an end to itself), but that they are also harmonized amongst themselves and that no area which should be regulated is forgotten. There must also be delineation of the law to ascertain where IT&T legislation starts and ends, and what is the specific liability of people who are involved in the legislative activity.

We are prepared to offer one possible legislative structure which is based on the need to establish principles that are considered fundamental in democratic societies and have long since not been the object of debate, as well as the need to regulate specific areas of activity. We believe in a certain principle of minimalism which says that a new law should not be created if the res-pective principle is enshrined in other laws (although supplements to existing laws may be necessary when new areas of potential regulation appear). Moreover, there is no need for regulation in sectors which operate successfully without it, which do not endanger people's health, national security or public order, and which do not bear the threat of financial losses. Further, we believe that there is no need at this time to regulate issues which in the "old" democracies are still the object of theoretical discussion among lawyers (legal status for electronic documents, for example, or "rules of the game" for global networks). In other words, we feel that the minimum requirements must be met, but no more than that.

What could be the main principles? This is not meant to be an exhaustive list, but some of the principles might be the following:

1) Each member of society has the right to be informed. Therefore, information from the state's institutions of authority and governance must be generally available. The accessibility can be limited only by law.

2) Information about individuals may not be disclosed if the disclosure would cause any type of harm to the respective individual. The permissible level of disclosure and the order for disclosure are set out in the law.

3) Intellectual work must be facilitated, ensuring people who are active in this area with sufficient and stimulating compensation.

One area which certainly needs regulation is the telecommunications sector, which is irreversibly linked to the field of information technologies. In keeping with the latest trends in the world, as well as with various EU directives, Latvia should repeal the existing monopoly on basic telecommunications services; this would require radical amendments to the law on telecommunications. It is difficult to say with any certainty how that would affect the modernization of telecommunications in rural areas, which is still continuing.

In several countries, including Lithuania and Slovakia, there are laws to regulate the establishment and operation of information systems of national importance (in some places these are called state registers), but such legislation by no means exists everywhere. For the aforementioned reason of minimalism, Latvia must see whether a new law is really needed in this area, or whether lower-level normative acts might not suffice.

In sum, we can conclude that there is no obvious need for law-based regulation in the IT&T sector, except in that the aforementioned three principles must be enshrined in law, and the monopoly in the

telecommunications sector may have to be repealed. If this conclusion is not overly hasty, then we feel that work on laws in the IT&T sector can be completed by the end of 1998.

## EXTERNAL IMPRESSIONS

At the end of 1997, France became the last of the 15 EU member countries to ratify the association agreement between Latvia and the EU. The Latvian government is planning to join a number of other international institutions soon (joining the World Trade Organization and signing the TRIPS Agreement, the WIPO Treaty, and the Convention for the Protection of Individuals with regard to Automatic Processing of Personal Data). All of these agreements will have a serious impact on the IT&T sector, but they do not provide for any major new principles or spheres which need regulation. All that Latvia need do is to establish its new laws in concert with the aforementioned agreements and to make the necessary amendments to existing laws.

## THE CURRENT SITUATION AND FUTURE DEVELOPMENTS

Presently (at the beginning of 1998), the situation with IT&T legislation is as follows:

- The law on informatics has been circulated among ministries for their approval. The law consists of two parts. The first specifies the distribution of authority among state institutions in the field of informatics, while the second addresses the issue of accessibility of state information. As we mentioned before, the law on state secrets, which limits overall accessibility, is already in force. The draft legislation was authored by a working group under the auspices of the Ministry of Transportation.
- It is expected that legislation on the protection of personal data will be elaborated by mid-1998. The work is being done by the same Transportation Ministry working group.
- Ministries are also reviewing proposed amendments to the law on copyright and related rights, which were authored by a working group at the Ministry of Culture. The draft includes various elements which are required by EU directives. The law will apply to compu-ter programs.
- The same Culture Ministry working group is to develop a draft law on protection of data bases by mid-1998.
- A working group at the Ministry of Justice has elaborated proposed amendments to the criminal code which would set up a system of serious sanctions for computer-related crimes. The criminal code already contains strict sanctions in the case of copyright violations. Similar amendments will also be needed in the administrative code.
- Negotiations are proceeding, albeit so far without results, about the creation of proper conditions to amend the law on telecommunications.

In sum, we can predict with some certainty that the present Parliament, which will be in office until November 1998, does have the possibility of completing work on legislation in the IT&T sector, ascertaining simultaneously that the laws are in concert with the requirements of the EU.

**Riga Information Technology Institute**
**Skanstes iela 13, LV 1013, Rīga, Latvia**
**Tel.: +371 2 374764**
**Fax: +371 7 821457**
**E-mail: juris.borzovs@dati.lv**

# Overcoming the Gap Between Latvian IT&T Legislation and the Laws of the European Communities

**Dr. Juris Borzovs Director, Riga Information Technology Institute;**
**Ilgvars Imša, Head of the Legal Department, DATI, Latvia**

***The authors will present a comprehensive review of laws and other normative documents related to IT and telecommunications which Latvia must adopt or revise in order to conform to EU requirements. Their review will be included in the Latvian National IT Program that is to be unveiled in October, 1997.***

A comprehensive review of laws and other normative documents in the field of information technology and telecommunications which Latvia must adopt or revise in order to conform to requirements of the European Union is currently being prepared. This review will be included in the Latvian National IT Program which is expected to be developed by October of 1997.

The goal of this paper is to take a look at the norms which directly or indirectly affect the field of information technology and telecommunications (IT&T) in Latvia; to provide a look at the process whereby these norms are being adopted in Latvia, pointing to the tasks which have already been fulfilled in bringing national law into line with EU directives; and to provide a comparative analysis from the perspective of national law and from the perspective of international rights.

This paper is not intended to be a comprehensive analysis of Latvian law in the field of IT&T, nor a full analysis of the extent to which this legislation does or does not correspond to European standards. Rather, the authors intend to reflect the problems which are faced by those who are seeking to direct this process and to answer questions about the process if possible.

In working on this paper, the authors accessed sources of information concerning national law, as well as international legal documents and commentaries on same.

When formulating tasks which must be performed in the area of Latvian law in the field of IT&T, one must devote specific attention to obligations which Latvia has accrued by virtue of various agreements that it has signed over the last several years with the European Union, the United States and Switzerland or that it is planning to sign with other countries.



## LAW IN THE FIELD OF INFORMATION TECHNOLOGY

IT is a new, unique and not fully recognized phenomenon in the world, one which knocks down any number of borders and boundaries. IT allows one to open oneself to the entire world, but this leads to problems which did not exist before. One of these problems is that the world has been opened on equal terms for everyone, including people who are irresponsible, negligent or even possessed of criminal intent. Accessed by the wrong minds, computer systems and the information which is stored in them can become a weapon or a victim of attack. We can mention just a few types of so-called 'computer crime':[1]

- Unauthorized access to information which is stored in a computer;
- Placement of 'logical bombs' into computer programs (these 'explode' under certain pre-determined circumstances, completely or partially knocking out the respective computer);

- Development and distribution of computer viruses;
- Criminal neglect in designing, manufacturing or using computer systems –something that in some instances has led to very serious consequences;
- Forgery of computer information;
- Theft of computer information.

This certainly suggests that there is an urgent need for laws in Latvia which would help to protect the system against such crimes. Even if laws to prohibit the aforementioned activities were in place, however, Latvia would also have to create a special executive structure –a 'data protection police', if you will –which would work on discovering and proving these kinds of violations of the law.

One of the most important elements in an IT&T protection system is the enforcement of the corresponding rights, and this is a process that is directly associated with the work of a country's court system. One must conclude, alas, that this segment of an IT&T protection system is not in place in Latvia for a variety of reasons (delays in the implementation of judicial reforms, a shortage of adequately specialized judges and attorneys, shortcomings in the law, etc.).

One cannot say, however, that there are no laws in Latvia to govern matters in the field of information technology.[2] The following laws are more or less directly associated with this field:

- The law on copyright and neighboring rights

This law3 addresses the issue of rights to use computer programs, and in this respect it is of critical importance in the field of information technology. The law states that rights to use a program must be purchased and that programs cannot simply be copied from another source. Evidence of a lawful purchase is provided by the respective license that is issued by the author of the program. Each program on each computer is supposed to have its own license, according to the law, the exception being so-called site or volume licenses. The issue of copyright when a program has been created by a salaried employee is addressed in the law, but in a way that is diametrically opposite to the way in which the issue is handled elsewhere in the world, including in a European Union directive.

The law provides for a compensation of losses in the event of the violation of copyright, including compensation for lost profits. The courts may order the confiscation or destruction of infringing copies of a computer program, and equipment and materials which have been used to copy the program illegally may be turned over to the ownership of the victim in the case to compensate for losses caused. The equipment and materials may also be donated to charitable causes, or a sum equal to the value of the equipment can be turned over to the national budget.

- The law on patents

The law [4] protects the rights of inventors, but it is of little significance in Latvia, as there is unlikely to be any major electronics industry in the country for some time to come, and computer programs are not protected by the patents law (which is not the case in the United States).

- The law on telecommunications

The law [5] has a secondary effect in the IT field. Its main purpose is to specify the rights of telecommunications organizations in Latvia and to govern all types of telecommunications activities in the country.

- The criminal code

The Latvian criminal code has several clauses that are of importance in this area. Paragraph 1366 specifies punishment for violation of copyright, neighboring rights and patent rights (the sanctions can range from a fine equal to 20 minimum monthly salaries to six years of imprisonment with confiscation of property). Paragraph 1327 sets our punishment for violation of the confidentiality of personal correspondence, of information transmitted via the country's telecommunications networks, or of information or programs intended for electronic data processing.

- The law on the Business Register of the Republic of Latvia

Peculiar as it may seem, the law [8] contains not a single word on the question of whether this register should be based on a computerized system of data. The law permits the disclosure of the names of a com-

pany's council or board members to any requesting party, but less proper under the principles of privacy rights is the fact that the law also permits the disclosure of the addresses of such people.

- The law on the Population Register

The law states that the register is an automated system for registration of the residents of Latvia, but there is nothing about regulations to govern the processing of data in this computerized register.

Along with certain shortcomings in existing laws [9], there are certain other, urgently needed laws which have not even been adopted in Latvia:

- A law on freedom of information;
- A law on protection of personal data;
- A law (or regulation) on the granting of legal authority to electronic documents;
- A law (or an amendment to the criminal code) to ban unauthorized access to information that is kept in computers (Paragraph 132 of the existing criminal code provides for a laughable fine of a sum equal to five minimum monthly salaries for breaking into a computer system if this has not been done with greedy intent and has not been done by an official person).

There is also a need to bring greater legal order to the system by which personal registers in Latvia are established and maintained. It is difficult to imagine how this process can continue to operate without a 'data police' or some similar institution.

## LEGISLATION ON THE PROTECTION OF INTELLECTUAL PROPERTY

When one analyzes Latvia's international obligations, which, among other things, are accrued within the context of every bilateral trade agreement to which the country is party, one finds that in some respects these obligations are not consistent from agreement to agreement (either in terms of their content or in terms of the time frame for their implementation). One cannot conclude from this, however, that these obligations (at least in the sphere of the protection of intellectual property) can be artificially divided into groups, each group to be fulfilled independently from all the others. This approach would clearly not work, because all of the obligations which emerge from Latvia's bilateral trade agreements are directly connected.

Latvia has, as is well known, begun the process of joining the World Trade Organization. Part of this process will involve acceptance of a covenant on trade-related intellectual property rights. Given this fact, as well as the fact that Latvia's treaties with the European Union include mentions of various WTO rules, including the aforementioned covenant, Latvia will do well to consider upcoming international obligations when it thinks about steps to take in the area of protecting intellectual property rights.

## EXISTING LAWS IN THE FIELD OF COPYRIGHT

One law has been adopted in Latvia in the field of copyright protection –the law on copyright and neighboring rights, which was adopted on May 11, 1993. During debate of the respective legislation, the World Intellectual Property Organization was asked to pass judgment on its basic provisions. The organization's assessment of the law was positive, which means that the law generally corresponds to the basic rules of EU legislation. Despite this fact, however, Latvia needs to take a look at the extent to which certain provisions in the law correspond to the aforementioned WTO covenant on trade-related intellectual property rights, as well as to an agreement which Latvia has with the United States concerning the protection of rights, especially in the area of copyright violations.

## LEGISLATIVE RESOURCES OF THE EUROPEAN UNION

The EU has the following legislative resources to implement its policy of harmonizing national legislation among EU member states:

Regulatory documents, which automatically take on the force of law in EU countries without any specific regulations or laws for their implementation;

EU directives, which do not automatically take on the force of law in member states but which are to be implemented in national law within a specified period of time. The effectiveness of this process is dependent on the extent to which each directive is precise and complete;

EU rulings, which usually concern specific subjects and are not all-encompassing;

EU recommendations, which are usually not binding and which are prepared by the European Commission or European Council with the purpose of setting out policy goals that are recommended to the EU member states;

EU opinions and resolutions, which are usually non-binding European Commission documents.

The EU has accepted several directives with the aim of eliminating shortcomings and differences in the national laws of member states in the field of information technology. One example is 96/9/EC March 1996 on the Legal Protection of Databases.[10] This document regulates the protection of any data base that has been created by a specific author, doing so from the perspective of copyright considerations. Within the context of this directive, a data base is an independent compilation of work that has been created systematically or methodically and the creation of which has required a certain intellectual investment by the author. A compilation of musical compositions by various authors, or by a single author taken from several sound recordings, would not be considered a data base under the terms of this law, because the compilation does not involve any intellectual investment.

An EU recommendation on the protection of personal data [11] specifies that if a data base contains private information about individuals, then the holder of the data base must ensure the protection of the individuals' private lives. In order to provide for an appropriate method of doing this, a whole new area of rights, which is known as data protection, has been created.

A directive will deal with the processing of personal data in the public and private sector, exempting from regulation such data as are processed in institutions that are not subject to EU regulation (e.g., institutions that deal with national defense).

Recommendations will set out the obligations of people who control data processing operations. Specific subjects that will be addressed include data quality and technical security, as well as conditions under which data processing can take place lawfully. In certain instances, the agreement of the persons whose data are being processed will be required, and such persons will have the right to be informed about the process and to access the data. They will have the right to request that incorrect information be corrected, and in some instances they will be able to demand that the data not be processed at all. In working on its own law on the protection of private data, Latvia should use the European Council's 1981 Convention for the Protection of Individuals With Regard to Automatic Processing of Data, as well as other regulations which EU member countries have adopted in this regard.

Another document which should be noted is the directive 91/250/EC May 1991 on the Protection of Computer Programs, [12] which addresses traditional issues of copyright protection. (In Latvia this function is filled by the law on copyright and neighboring rights.) The directive specifies that the owner of copyright has exclusive rights which include control over the duplication of his or her computer program by any means and in any form, the right to supervise the translation, duplication or alteration of the computer program, and the right to control the public distribution of the program and the leasing of the program's original version or any copies. The directive was approved by the Council of the European Communities in May 1991, and in several respects it is a unique piece of legislation:

1) It is the first EU document which is meant to harmonize the copyright laws of member countries;
2) It is the first time that access to interface information is specifically addressed in legislation;
3) The adoption of the document involved an unprecedented flurry of American-style lobbyism in Europe. [13]

The directive also regulates the following issues:

1) The inter-operability of computer programs and its range and form;
2) The protection of interface;

3) The maintenance of software;
4) Piracy.

The directive has specific regulations on who may carry out decompliation, in what events, for what purpose and to what extent.

## EU LEGISLATION IN THE FIELD OF TELECOMMUNICATIONS

Regulation of telecommunications by the European Union has a longer history than does the regulation of IT, and the process has involved greater experience and more thought. The goal of the legislation is to provide a legal basis for the successful and ongoing operations of the telecommunications sector and related sectors, achieving uninterrupted, successive and stable development of the sector. The demands which the EU has made vis-a-vis Latvia in this respect are many and concrete in nature. The world's experience indicates that the operations of this sector require basic laws on the basis of which regulations, normative acts and legal demands can be made. The main characteristic of such basic laws is their stability and constancy over at least ten years, which allows organizations to plan, forecast and develop their operations. In Latvia these functions are filled by the law on telecommunications and the law on the country's postal service. [14] The interests of the radio and television sector are affected by the law on the electronic mass media.

In terms of Latvian cooperation with the European Union, the EU has specified several priorities in this area of activity which should be noted:

- The establishment and implementation of legal acts and procedures in the field of telecommunications;
- The modernization of the respective data network.

On the basis of this, the EU has issued a series of directives which Latvia must observe when developing its own national laws. Paragraph 85 on the association agreement which Latvia has with the European Communities, for example, specifies that telecommunications, postal services and television and radio broadcasts are areas in which Latvia must harmonize its laws with the following EU directives:

- 86/361/EEC (OJ L217/21-5/8/86) – On the mutual recognition of initial telecommunications terminal systems.
- 87/372/EEC (OJ L 196/85-17/7/87) – On frequency bands which are to be reserved for the implementation of pan-European public cell surface mobile digital communications in the European Community.
- 88/301/EEC (OJ L 131/73-27/5/88) – On competition in the market for telecommunications terminals.
- 90/387/EEC (OJ O192/1-24/7/90) – On the establishment of domestic markets in the sphere of telecommunications services, implementing the terms of an open network.
- 90/388/EEC (OJ L192/10-24/7/90) – On competition in the market for telecommunications services.
- 90/544/EEC (OJ L 310/28-9/11/90) – On frequency bands which are to be reserved for the implementation of a pan-European public surface radio paging system in the Community.
- 91/263/EEC (OJ L128/1-13/5/91) – On harmonization of legislation of member countries with respect to telecommunications terminal systems, including recognition of their mutual conformity.
- 91/287/EEC (OJ L144/45-8/6/91) – On frequency bands which are to be reserved for implementation of a European digital wireless telecommunications (DECT) system in the Community.
- 92/44/EEC (OJ L165/27-19/6/92) – On the application of the principles of an open network to leased lines.
- 89/552/EEC (OJ L298-17/10/89) – On harmonization of the terms of member country laws, regulations and administrative instructions with respect to television broadcasts.

## CONCLUSIONS

Participation in the European information society will mean not only rights for Latvia. The road to this society is full of obligations and work to be done.

The authors would like to thank the director of the Department of Informatics of the Ministry of Transportation of the Republic of Latvia, Dr. Andris Virtmanis, for his assistance in the preparation of this document.

## REFERENCES

1. Baturin, J.M. 'Komputernaja prestupnostj I kompjuternaja bezopastnostj.' Moscow, Juriditeskaja literatura (1991).

2. See Borzovs, J. 'Information Technology in Latvia: Laws and Standards', Baltic IT Review, No. 3, 1996.

3. 'Par autortiesībām un blakustiesībām' (On Copyright and Ancillary Rights), Ziņotājs, 10 June 1993, No. 23.

4. 'Latvijas Republikas patentu likums' (The Patents Law of the Republic of Latvia), Vēstnesis, 19 April 1995, No. 10.

5. 'Latvijas Republikas likums Par telekomunikācijām' (The Law of the Republic of Latvia on Telecommunications), Vēstnesis, 19 April 1995, No. 60.

6. 'Latvijas Krimināłkodekss' (The Latvian Criminal Code), Paragraph 136.

7. Ibid., Paragraph 132.

8. 'Latvijas Republikas likums Par Latvijas Republikas Uzņēmumu reģistru' (The Law of the Republic of Latvia on the Business Register of the Republic of Latvia), Vēstnesis, 21 October 1995, No. 162.

9. See, for example, Borzovs, J. 'Gatavojamies pirmajai tiesas prāvai par datorprogrammu autortiesību pārkāpumiem' (We are preparing for the first court case concerning violation of computer program copyrights), Datortehnika, No. 1, 1993, p. 31.

10. EC Directive No. 96/250, O.J.L 122/42 (1996).

11. White Paper, 14th chapter EC COM (95/63), final version, Brussels, 3 May 1995.

12. EC Directive 91/250/EC May 1991 on the Protection of Computer Programs, Official Journal 1991, L122/42.

13. 'International Protection of Intellectual Property', Pearson Professional Subscriptions Department, EU/3.

14. 'Pasta Likums' (The Law on the Postal Service), Vēstnesis, 31 May 1994.Baltic Government Information Exchange and a common information infrastructure.

# STOCKHOLM

BEAUTY ON WATER

The 8th European Conference on
INFORMATION SYSTEMS SECURITY, CONTROL
& AUDIT—the impact on management
September 6-8 1993, City Conference Center, Norra Latin



EDP AUDITORS ASSOCIATION, INC.®
FOUNDED 1969

Stockholm Chapter

# Is there a Need for Software Quality Assurance in Latvia?

**Juris Borzovs, Dr.Comp.Sci.**

**University of Latvia, Docent**
**Software House Riga, Software Quality Manager**

**1. Pecularity of Latvia**
**2. Software Quality as a Fuzzy Entity**
**3. Standard as a Shield for Everybody**
**4. Draft Proposal for Taxonomy of Software Tools and Aids**

## 1.

**Latvia is rather peculiar and therefore interesting region also from software quality reason. From one side, Latvian scientists are pretty well-known due to their research in automating of software testing and specification [1-3]. From other side, there is lack of general standards and procedures in program development. Thus there is no reason to speak on stable engineering software quality assurance tradition.**

**However, along with dramatic change of economy structure both external and internal competition are raising fast. Internal competition increases mainly due to economy slow down and decrese of payable consumers. External competition increses due to trend of several local software firms to fulfill foreign orders. The most interesting example is Software House Riga, private shareholders company with over 400 software specialists comprising a bulk of the best professionals mainly from universities. In both cases quality is a decisive factor.**

**Quality is not only decisive but also expensive entity. As far as we know only Software House Riga has alredy established specific qulity group this year in order to support high-quality production.**

## 2.

**If any entity can not be mesured it almost certainly can not be controlled. What about software quality? What do the classics say [4] ?**

**A software product is delivered to you. Is it good or bad? Your interest may be casual: three programs are available to calculate lunar transfer orbits and you would like to work with the best one. Your interest may be more intense: the product represents all, or part, of a project you are**

managing, or a product you are buying. The goal in each case is the same - to assess the quality of the product. But your interests and and mesuring criteria are completely different. If you are trying to pick the best program from a group, your interests might center on portability - can I use it here - or modifiability, understandability, or accuracy. If the product is part of a larger project, then your interests may center on smooth controlled evolution. Is the product responsive to requirements? Is it complete? Are project standards adhered to?

There can be no single quality measure. The process of assessing the quality of a software product begins when specific characteristics and certain metrics are selected. Some of the metrics may require added information to be supplied on standards, priorities,etc. The product is mesured by the procedures of each metric, sometimes by an automated program or device, in other cases by a human. The metric scores are recorded. The output from the metrics can be a numerical score (percentage completed, percentage of error in test cases,etc.), or a qualitative human judgement in one of two forms: a numerical rating or a yes/no decision. the scors may be combined into a figure of merit. However, since the metrics as yet do not measure all components of a characteristic, the figure of merit will be more suggestive than conclusive or prescriptive. More likely, the individual metric scores are examined for deviations from nominal or expected results. Anomalies will be detected. Patterns will emerge and can be classified. Serious deficiencies will be examined and corrected, or often disregarded when it is found that the criteria are not applicable to the product at hand. For example, low ratings in understandability may be an acceptable defect in highly efficient real-time code.

To summarize:
1) The quality of a software product varies with the needs and priorities of the prospective user.
2) There is, therefore, no single metric which can give a universally useful rating of software quality.
3) At best, a prospective user could receive a useful rating by furnishing the rating system with a thorough set of checklists and priorities.
4) Even so, since the metrics are not exhaustive, the resulting overall rating would be more suggestive than conclusive or prescriptive.
5) Therefore, the best use for the metrics at this point is as individual anomaly indicators, to be used as guides to software development, acquisition, and maintenance.

Surprisingly, how contemporarily these judgements sound, even after decades.

3.

Persons engaged in software development and usage, naturally, have different, sometimes even contradicting interests. CONSUMER spending his/her money wants to obtain completely reliable, adhering to his/her needs and, in addition, inexpensive product. PROGRAMMER does not want to produce quite bad, compromising product, although ingenious lazyness and permanent lack of time stimulate him/her not to do all that is possible in principle. MANAGER also is pressed by different factors, for example, usual deficit of time and money, a wish to satisfy CONSUMER and to get

maximum from PROGRAMMER. MANAGER bears on responsibility of product quality for CONSUMER and requires internal responsibility on product from PROGRAMMER.

The situation is made quite piquant by well-known fundamental results of theory of algorithms stating that no nontrivial program property is algorithmically decidable [e.g., 5]. Thus quite simple but not trivial program property "to be correct" (and many similar) is not decidable in principle. If so, what can be required both from PROGRAMMER and MANAGER? Theoretically - nothing.

From other side - even very skilled, talanted and honest programmer if ordered a product of maximum quality should improve and improve it infinitely (because can not afford 100% quality).

What exit possible? Compromise needed. And STANDARD is a compromise between wishes and possibilities. STANDARD is nothing more than officially accepted convention between professionals on minimum needed requirements regarding product and its development process. STANDARD itself does not guarantee anything 100% but, anyway, it is concentrated experience proved to be sufficient to provide quality product.

So, if PROGRAMMER has obeyed every STANDARD requirement MANAGER has no right to reproach, accuse him/her, even in case of software failure or crash. In the same way, CONSUMER has no right to accuse MANAGER. There was an accident and nobody has to be accused to the court.

Well, but what a shield CONSUMER and MANAGER have in front of PROGRAMMER? The same STANDARD! The STANDARD protects them from product below admitted level of quality produced outside of admitted development process.

Consequently, everybody is essentially interested in existance of STANDARD and in its obeying.

4.

For an overall viewpoint, the creation of Software Engineering (SE) standards has been an evolutionary process. Let us examine briefly the American SE Standards system [6]. As soon as this one was the only published Western system available in Latvia we have begun to form a fundament on this system.

Recall that first came common vocabulary (IEEE Std 729-1983) and initial framework of relationship (IEEE Std 730-1984). Only years later taxonomy of SE standards (IEEE Std 1002-1987) emerged providing the audience with systematic view and outline of standard types and application areas, somewhat very like table of chemical elements devised hundred years ago. This taxonomy clearly shows needed directions of design and implementation of new standards. The first is to continue the definition and expansion of the requirements contained in Std 730 and Std 1002. The second is to move from Standards and Recommended Practices to Guides. The third goes from a

product standard (for example, testing) to a process standard (for example, testing). The last is the expansion of the terminology, both to include additional terms themselves and to define associate metrics (for example, productivity metrics).

However, there are several problems with nearly ever standards systems to be implemented in everydays manufacturing process. The first one is incompleteness because there is certain several years period between stating of a need and producing a standard. We can mention an urgent need for, at least, following standards in the American system: Standard for the Software Life Cycle Processes, Standard for Baselines, Standard for Concept Design, Standard for Software Maintenance, Guide for Third Party Software Acquisition, General Coding Standard,etc.

The second problem is a need for development process complete or, at least, partial automation. Practitioners need standards application examples, application manuals and tutorials, automated or semiautomated supporting tools,etc. Existing standards give only a framework, fundament and directions for specific implementation.Thus we see that the Taxonomy Std 1002 could be enhanced by addition of software tools and aids partition giving directions for systematic and disciplined approach to Quality System development. We demonstrate below only a small fragment of the partition as an example.

| Type of Tool or Aid | |
|---|---|
| Aid ("for eyes") | Tool ("for hands") |
| Product standard | Templates or spare parts |
| Product example | Repository of subproducts |
| Process standard | Repository and subproduct evaluation |
| Process guide | Repository, evaluation, partially automated production |
| Process example | Automated production tool |

References

1.J.M.Barzdin, J.J.Bicevskis, and A.A.Kalninsh. Automatic Construction of Complete Sample Systems._ Proc.IFIP Congress 1977._North-Holland,1977,pp.57-62.

2.J.Bicevskis, J.Borzovs, U.Straujums, A.Zarins, and E.F.Miller,jr. IEEE Transactions on Software Engineering,SE-5,No.1,1979,pp.60-66.

3.J.Barzdins, D.Bjorner (Eds.) Baltic Computer Science._Lecture Notes in Computer Science,No.502,Springer-Verlag,1991,pp.286-432.

4.B.W.Boehm, J.R.Brown, H.Kaspar, M.Lipow, G.J.MacLeod, M.J.Meritt. Characteristics of Software Quality._North-Holland,1978.

5.H.Rogers,jr. Theory of Recursive Functions and Effective Computability._McGraw-Hill,1967.

6.IEEE Standards Collection SOFTWARE ENGINEERING,1993 Edition._Institute of Electrical and Electronics Engineers,1993.

# Harmonization of Latvian and International IT Standards: An Enormous Effort is Needed

**Solveiga Bērziņa and Dr. Juris Borzovs, Rīga Information Technology Institute**

***As a would-be member state of the European Union, Latvia will be required to join the European standards organizations CEN and CENELEC as a full member. In order to do this, Latvia will have to adopt all European standards as national ones and withdraw any national standards that do not conform to the European ones. A pre-condition for joining CEN and CENELEC is that the respective country must have at least 70% of their standards adopted at the national level. In some cases, this can mean the adoption of several thousand (!) standards, and that, of course, cannot be accomplished in a short time or without very serious resources.***



## THE CURRENT SITUATION

Possible strategies for the development of Latvian IT standards have been elaborated several times [1-4] before. Similar information is also available about Estonia and Lithuania [5-6]. Until now, the adoption of international standards and the harmonization of national standards with international ones has fully been the provenance of the Latvian government. Sad to say, the government has virtually ignored the issue of standardization, leaving any initiative in this area to individuals, and failing to allocate any state money even for the support of the official Latvian National Standardization and Metrology Center (LNSMC), to say nothing of more unofficial pursuits [3].

The strategic situation has now changed fundamentally, because Latvia has now signed an association agreement with the European Union. Of interest here is the section in the agreement which addresses "industrial standards and conformity assessment". The agreement requires Latvia to eliminate differences in standards, technical regulations and conformity assessment procedures. Although joining the EU is not something that will happen next year, Latvia does have to bear in mind that EU member states are required to join the European standardization organizations CEN and CENELEC. A requirement for members is that each country adopt European standards at the national level and that it withdraw any non-conforming national standards. Indeed, 70% compliance is required as a pre-condition for membership in CEN and CENELEC. For some countries this means the adoption of thousands of standards, and this cannot be done quickly or cheaply. Some sources report that Sweden spent the equivalent of US $7 million in the process of standards adaptation.

Because of the severe scarcity of resources in Latvia, it will never be possible to adapt a great number of voluminous standards by translating them into the Latvian language. This would not, in fact, be expedient, because most standards are not at all widely used. In most cases, the so-called "cover page" approach should be applied: the first page of the standard is an abstract in Latvian, while the rest of the text is in one of the working languages of the European standardization organizations –German, French or (most often) English. The bulk of standards should be adopted without any translation at all, with the Information Technology Standardization Technical Committee (ITSTK) recommending and the LNSMC

*Table 1. EN Information Technology Standards*

| Total number | | In possession of LNSMC | | Still needed | |
|---|---|---|---|---|---|
| Number | Pages | Number | Pages | Number | Pages |
| 163 | 10,029 | 161 | 9,948 | 2 | 81 |

announcing adoption of each standard. However, the ITSTK firmly believes that the original text of each standard must be made available in Latvia before it can be adopted, and all terminology standards must be translated.

The adoption of ISO, IEC and other standards is not mandatory for Latvia at this time, but this should be done nevertheless, because European standards do not always deal with all relevant issues. In short, there is an enormous job to be done. Someone has to read each standard that is to be adopted and select the method of adaptation (full translation, cover page, announcement). This job cannot be done mechanically, because without careful analysis, Latvia could end up in a situation where standards from different systems (i.e., one from the ISO and the other from CEN) end up contradicting each other. Such inconsistencies should be avoided in the Latvian standards system, or at least there should be some reconciliation in the form of guidelines (as in [1]). Furthermore, there will be cases when standards from one system (for example, the IEEE) will have to be applied simply because other organizations, such as the ISO, the IEC, the CEN and CENELEC have not yet standardized the respective sphere. At that point consistency will be a much more serious issue.

*Table 2. ISO Information Technology Standards*

| Total number | | In possession of LNSMC (since 1995) | | Still needed | |
|---|---|---|---|---|---|
| Number | Pages | Number | Pages | Number | Pages |
| 1,062 | 53,500* | 493 | 26,452 | 569 | 27,050* |

* Approximate

*Table 3. ISO Terminological Standards*

| Standard identificator | Title | Pages | Available in LNSMC |
|---|---|---|---|
| ISO/IEC 2382-1 | IT-Vocabulary-Fundamental terms | | no |
| ISO 2382-2 | Data processing (DP)-Vocabulary (V)-Arithmetic and logic operations | | no |
| ISO 2382-3 | Information processing systems (IPS)-V-Equipment technology | | no |
| ISO 2382-4 | IPS-V-Organization of data | | no |
| ISO 2382-5 | IPS-V-Representation of data | | no |
| ISO 2382-6 | IPS-V-Preparation and handling of data | | no |
| ISO 2382-7 | IT-V-Computer programming | | no |
| ISO 2382-8 | IPS-V-Control, integrity and security | | no |
| ISO/IEC 2382-9 | DP-V-Data communication | 43 | yes |
| ISO 2382-10 | DP-V-Operating techniques and facilities | | no |
| ISO 2382-11 | IPS-V-Processing units | | no |
| ISO 2382-12 | IPS-V-Peripheral equipment | | no |
| ISO/IEC 2382-13 | DP-V-Computer graphics | 42 | yes |
| ISO 2382-14 | DP-V-Reliability, maintenance and availability | | no |
| ISO 2382-15 | DP-V-Programming languages | | no |
| ISO/IEC 2382-16 | DP-V-Information theory | 19 | yes |
| ISO/IEC 2382-17 | IT-V-Databases | 33 | yes |
| ISO 2382-18 | IPS-V-Distributed data processing | | no |
| ISO 2382-19 | IPS-V-Analog computing | | no |
| ISO 2382-20 | IT-V-System development | | no |
| ISO 2382-21 | DP-V-Interfaces between process computer systems and technical processes | | no |
| ISO 2382-22 | IPS-V-Calculators | | no |
| ISO/IEC 2382-23 | IT-V-Text processing | 43 | yes |
| ISO/IEC 2382-24 | IT-V-Computer-integrated manufacturing | 13 | yes |
| ISO/IEC 2382-25 | IT-V-Local area networks | 44 | yes |
| ISO/IEC 2382-26 | IT-V-Open Systems Interconnection | 34 | yes |
| ISO/IEC 2382-27 | IT-V-Office automation | 26 | yes |
| ISO/IEC 2382-28 | IT-V-Artificial intelligence | 28 | yes |

In total, there is a need to purchase approximately 60 ISO standards, and this might cost approximately US $3,000.

## MAJOR SOURCES FOR IT STANDARDS

At this point our research has basically just scratched the surface and is based on information from the LNSMC, as well as five years of systematic participation in the standardization activities of the Institute of Electrical and Electronics Engineers. We have not yet gotten to the collection of the Latvian Patents Technical Library (which collected ISO, GOST and other standards from Moscow until 1990 and which may be receiving other standards now). Thanks to methodological assistance from the German standardiza-

*Table 4. Other needed IT standards*

| Body | Urgently needed | Useful |
|---|---|---|
| ECMA | 11 | 14 |
| IEC | 7 | 10 |
| ISO | 18 | 12 |
| ANSI | 1 | 5 |
| ASTM | 4 | 11 |
| ISO/IEC | 15 | 10 |
| ETSI | 2 | 1 |
| Total | 58 | 63 |

tion institution DIN, we have received an enormous catalogue of standards [7], which enumerates no fewer than some 2,500 IT standards. There is also a data base of standards on CD. We have concluded that the most significant standardization organizations in the IT sphere are the international organizations ISO, IEC, ECMA, ITU (CCITT), ETSI, IEEE, ASTM, CEPT, CISPR and CIE, as well as the national ANSI. We should note that a majority of standards adopted by DIN have been kept in the English language, which demonstrates that even a large and wealthy country like Germany does not see any need to engage in mass translation of standards.

Because the authors of this article are themselves involved in the standardization of terminology and software engineering, the following remarks will undoubtedly be a bit subjective and by no means are intended to deliver a final opinion on matters. We have, however, sought to provide an honest analysis of the need to adapt standards from other spheres of information technology.

## EUROPEAN STANDARDS

Thanks to the aforementioned DIN, the LNSMC has received the texts of virtually all of the European standards that must eventually be adopted in Latvia. There are approximately 4,000 in all. Unless we have made a serious misjudgment, then we find that a fairly small percentage of all European standards applies to the field of IT (see Table 1). More than half of these involve telecommunications and computer networks. Full translation would probably we warranted for the terminological standard (12 pages) and perhaps seven other standards which deal with measurement and systems safety and which should be available in Latvian. We see, therefore, that the harmonization of standards in the field of IT would not be a particularly resource-intensive process, and we can only guess about the sectors which will have to deal with the other 3,800 standards.

## ISO STANDARDS

ISO standards (Table 2) make up no less than 40% of the IT standards. Since Latvia has been an "O" member of the ISO since 1995, the LNSMC receives all new and revised standards. Over the course of two years, the LNSMC has received almost half of the existing ISO standards, and one can assume that by the turn of the millennium, without any major effort on our part, we will have access to all of them in Latvia. We feel, however, that terminological standards (Table 3), and those standards which are of particular need in system design (there are approximately 35 of them) should be purchased as soon as possible.

## IEEE STANDARDS

The IEEE is the international, professional organizations that works in the area of software engineering and terminology. Latvian software engineering standards have been developed on the basis of the IEEE standards. Of more than 800 IEEE standards, 211 involve IT (that amounts to some 30,000 pages). The authors have access to the organization's catalogue of standards [8], as well as a majority of software engineering standards, but they have not yet been made publicly available in Latvia. There is an urgent need to purchase at least the Standard Computer Dictionary with supplements (approximately 400 pages

at a price of some $300) and the Software Engineering Standards on CD-ROM ($600). Also of use would be the IEC Multilingual Dictionary of Electricity. Electronics and Telecommunications (15,000 terms, 7 languages, $200).

## OTHER STANDARDS

A look at the standards lists of the other aforementioned organizations, one gets the sense that they, too, have significant documents (Table 4) which should be obtained if possible.

## SUMMARY

This very preliminary research in the IT standards field has allowed us to estimate that at least 40 standards (mostly terminological) with a total of some 1,500 pages should be translated, while 200-300 others should be adopted by the cover page or announcement method. This job might involve approximately 300 person months of effort and a cost of some $500,000. The cost of purchasing the standards could range from $5,000 and $20,000.

## REFERENCES

1. Ādamsone A., Borzovs J., Ēvere R. and Lučkina, M. "Framework for Potential Latvian Software Standards", in Haav, H.M and Thalheim, B. (eds.). Databases and Information Systems: Proc. 2nd Int. Baltic Workshop. Tallinn (1996), pp. 105-112.

2. Borzovs, J. "Informācijas tehnoloīija Latvijā: likumi un standarti" (Information Technology in Latvia: Laws and Standards), in Latvija ceļā uz informācijas sabiedrību, Riga: Latvian Academic Library (1996), pp. 35-40.

3. Mētra, I. "IT Standardization Activities in Latvia", Baltic IT Review, No. 3, 1996, pp. 48-50.

4. Borzovs, J. "Information Technology in Latvia: Laws and Standards), Baltic IT Review, No. 3, 1996, pp. 44-47.

5. Agur, U. "The Situation with IT Standards in Estonia", Baltic IT Review, No. 3, 1996, pp. 32-34.

6. Tumasonis, V. and Stanislovaitis, A. "A Brief Review of Information Technology Standards, Standards Policy and Standards Legislation in the Republic of Lithuania", Baltic IT Review, No. 3, 1996, pp. 54-56.

7. DIN Catalogue of Technical Rules, Vol. 2: International Standards and Selected Collections of National Standards. Berlin, Vienna and Zurich: Beuth Verlag GmbH (1996), 1216 pp.

8. IEEE Standards Product Catalog, 1997.

# The Latvian Language and IT&T

**Dr. Juris Borzovs, Director, Riga Information Technology Institute**

**Dr. Guntis Fricnovičs, Deputy Director, Institute of Electronics and Computer Science, University of Latvia**

**Dr. Andrejs Spektors, Leading Researcher, Institute of Mathematics and Computer Science, University of Latvia**

***There is no longer any need for argumentation about the usage of the Latvian alphabet in computer environments. IBM, Microsoft and other, smaller companies include it into their products which are distributed in Latvia. Common issues concern user documentation in Latvian, etc. However, more and more often we realize that problems have arisen which have not been resolved for reasons that are entirely the fault of the Latvians themselves and not anybody else.***



## WHERE ARE WE NOW?

Destiny has fated us to live at the beginning of the Information Society. This phase of development of the IS has been characterized, on the one hand, by an enormous increase in the flow of information and, on the other hand, by the development of information and telecommunications technologies and the gradual convergence of both types of technology [1,2]. Most of this rapidly increasing information flow consists of information that is conveyed in natural languages. Development of computerized processing of information, as well as telecommunications technologies, have allowed us to access ever greater opportunities for complex information processing, as well as simultaneous transmission in various languages and in different forms of representation, including hypertext and multimedia.

Do these developmental tendencies have an effect on our Latvian language, and if so – what is this effect? What must we do to become involved in these developmental processes without sacrificing our language? Is that even possible? In order to find answers to the questions, we must look around and seek to understand the path which Europe is taking and the path which we ourselves are following.

In the European Union today, there are nine official languages, and a number of languages spoken by a few thousand people or several million people. According to the number of people who speak a language, we often speak of so-called 'big' and 'small' languages. Behind these languages we find thousands of years of culture which cannot be perceived as 'small' or 'big'. The languages are the bearers of these cultures. If the small languages are threatened, then so are the cultures and societies linked with them [2, K. Stroerup].

Europe believes that these threats can be escaped by purposely establishing a multilingual society and not by leaving these processes to the whims of market forces alone. Otherwise we may end up with a situation where just a few languages – or even just one language – remain as the bearers of information. There are languages which may be threatened by current market developments. Here we must remember the so-called Guttenberg effect: If no published works are printed in a language, then the language may disappear completely or remain at the conversational level. Languages which are not used in computers, by extension, are being threatened today. There is only one way to maintain a language in this process – information technology, electronic communication networks and language processing systems have to be developed for many languages at the same time [2, W. Teubert].

In order to preserve the Latvian language, in other words, we must use all of the opportunities that are afforded by present-day technologies to make links between the Latvian language and culture and other languages and cultures. That means that if we are to become involved in the development of Europe's

multilingual society, we must include in our National IT programme at least three basic tasks, and we must ensure that there is state support for the performance of these:

- The Latvian language texts must be processed by computers;
- Latvian language processing systems must be developed;
- Latvian terminology must be developed and standardized.

What is the current situation in the pursuit of these aims, and what activities in this area are being planned in the near future?

## THE LATVIAN LANGUAGE AND INFORMATION TECHNOLOGY: TWO VIEWS

The connections between the Latvian language and information technology can be considered from a variety of viewpoints, but two of these seem to be the most important. One concerns the manufacturing and distribution of a so-called 'Latvian computer', while the other deals with the supporting of the Latvian language as such.

Both issues are in fact dependent on the political doctrine which governs the use of languages in the Republic of Latvia for the foreseeable future. A proposal on a new national language law has been stuck in Parliament for some time, and it is difficult to predict with any certainty its final version. The primary aim of the current language law has been to return the Latvian language to the status of most-used language in the country, replacing Russian, which currently holds that status, at least in the area of unofficial discourse. The world is not limited to these two languages alone, however, and even more – nothing can be accomplished with either language in the countries of the European Union. The Latvian government must implement a long-term language education strategy, deciding first and foremost what languages must be studied in elementary school and what level must be achieved. We specifically stress the concept of achievement, rather than simply learning. We must stop the current process of producing functional illiterates who have studied as many as four languages but can actually function in no more than one of them. We feel that every graduate of Latvian elementary schools (to say nothing of high schools) must be able to use three or four languages actively. These must be the Latvian language (which is the official state language) so that the young person does not face discrimination in Latvia and feels himself or herself to be part of the country's culture; the English language, without which one cannot survive in Europe and in many other places in the world; the Russian language, which is the language of Latvia's largest minority and which opens doors to vast territories to our East; and, finally, the mother tongue for any young person who desires. (Even the authors disagree on this "4 language" issue.) The adoption of a long-term strategy like this one will require radical changes in the way languages are taught in Latvia, but that is not the theme of this paper.

## LATVIAN LANGUAGE USE IN COMPUTERS

If the English language were taught (successfully) from the very earliest grade levels, we might be able to make do without any 'Latvianization' of computer programs. This is not an issue of scorning the Latvian language or of attempting to shunt it aside. Rather, we are simply recognizing a bitter truth – that there are far too few Latvians to 'Latvianize' even a tiny share of what has already been developed for English language use and what will be developed in the future. There is an enormous difference between the need to 'see' and process Latvian texts in computers and the need to communicate with the computer in that language. Computers which are able to provide the latter of these functions can be said to be 'Latvianized'. There are rumors that one computer enthusiast has already programmed such a computer for himself.

With respect to text processing in the Latvian language, this issue has largely been resolved, at least with respect to the most widely used computer programs. Of course, any new computer program requires a certain amount of adaptation (most often this involves changing the code table), and this process is more difficult if the author of the program has not anticipated the use of various languages during the design phase. It can be expected that the issue of text processing in various languages will be resolved

completely in the foreseeable future, when the Unicode [3,4] 16-byte code table is implemented. Luckily, the Latvian alphabet has already been included in Unicode, so nothing additional will have to be done.

The issue of a 'Latvian computer' is something altogether different. The 'Latvianization' of user interface procedures requires much greater resources, even on occasions when the use of other languages has been anticipated from the very start of the program. The 'translation' of a medium-sized system can cost tens of thousands of dollars. Furthermore, a computer can be truly 'Latvian' only if the user interface of its operating system is also 'Latvianized', and this would involve the translation of enormous help files and diagnostic texts, as well as user handbooks. This would cost hundreds of thousands of dollars, to say nothing of enormous investments of time and labor. We feel that Latvians must be realistic about this situation and recognize that it may very well be true that there will never be mass-produced Latvian computers. A certain compromise might be reached by providing special Latvian-English keyboards for computers that are sold in Latvia [5]. We hear that there are no computers in Estonia which do not have keyboards appropriate for the use of that language.

## DEVELOPMENT OF LATVIAN LANGUAGE PROCESSING SYSTEMS AND LANGUAGE CORPORA

When we speak of the Latvian language in information technologies and telecommunications, we must recognize the overall trends which are prevailing in the development of these systems – that which has become known as the global village effect. What language is to be used for the exchange of information? How long will the Latvian language even be needed given these circumstances? Can we do anything to help our language survive in the world of the future? It seems that there can be only one solution: functions of information processing and translation must be turned over to computer equipment and intellectual programs to as full an extent as is possible [6]. The Institute of Mathematics and Computer Science of the University of Latvia has been researching the PC-based processing of Latvian language dictionaries and texts since 1988 [7].

If research on contemporary Latvian language issues is to continue, however, it will be very necessary to quickly introduce as many Latvian language texts as possible into computers and to mark them in correspondence to international standards. Only after this enormous piece of work is accomplished will we be able to talk of truly modern Latvian language research. Latvian language data banks could also serve as a basic resource for automated synthesis of knowledge [8], thus creating the pre-requisites for the development of artificial intelligence, including automated translation [9], in the Latvian language.

Specialists believe that further research in linguistics and computer linguistics is reasonable only if tens of millions of word usages are placed into the corresponding data bases [10, 11]. That is especially important when discussing inflected languages, where the inflections of each word must be taken into account. The largest computerized Latvian language resources in existence at this time are at the Artificial Intelligence Laboratory of the Institute of Mathematics and Computer Science; approximately 10 million units of word usage are contained there.

Over the next several years, specialists should create the necessary programming for automated development of a Latvian language computer fund. The development of programs that can optically recognize letters from the Latvian language is necessary so that a large quantity of Latvian language texts could be entered into computers.

Special research and programming tools are also needed to create and use parallel corpora [12]. Parallel corpora contain identical texts in two languages and are used as translation resources.

Specialists at the Artificial Intelligence Laboratory have developed a program for morpheme analysis (i.e., automated analysis of words which are encountered in a text [13]). The system provides correct responses in 90% of cases.

It is also necessary to develop methodologies, algorithms and programs that can establish and mark speech corpora. Right now there is only one data base – a collection of the pronunciations of proper

nouns in the Latvian language [14]. This data base must be supplemented with the pronunciation of additional morphemes. Sound models for the Latvian language must also be developed so that computerized speech can be generated from texts and so that individual speakers can be recognized [15] sufficiently to allow the computer to write down spoken words. No less important is the development of a state-of-the-art Computer Aided Language Learning (CALL) program for the Latvian language, as well as the placement of as many Latvian language instruction materials as possible on the World Wide Web. We must also provide for the electronic exchange of documents in the Latvian language, keeping in mind the European Union's Open Document Architecture (ODA) standards when doing so [17].

The overall purpose of these plans is to bring the computerized linguistic resources of the Latvian language closer to the level that prevails in countries of the European Union.

## LATVIAN TERMINOLOGY

The Terminology Division of the Latvian Language Institute has conducted fundamental research in the area of Latvian terminology [18], collecting several hundred thousand terms from various sectors. Work on Latvian terminology became particularly active after the restoration of Latvian independence and the introduction of the state language law. Cooperation has begun with the terminology research and information center in Vienna (IITF, INFOTERM), the International Organization for Unification of Terminological Neologisms (IOUTN), and the International Federation of Terminology Banks (IFTB). Joint research projects have been undertaken under the auspices of the COPERNICUS program [19]. Because of the very rapid development of information technologies, an IT sub-committee was established in 1992. More than 90 meetings of the sub-committee have been held, and approximately 4,000 terms in the fields of data processing, personal computers and computer networks have been recommended. An English-Russian-Latvian explanatory dictionary, Data Processing and Transmitting Systems, has been published, and a second dictionary, Personal Computers, has been prepared. A similar dictionary on the field of computer networks is being planned.

On the basis of all of the aforementioned, one may conclude that everything is fine in the field of Latvian terminology. Such a conclusion would be premature, however. As INFOTERM specialists have noted [1], terminology today is regarded as the representation of specialized knowledge at the level of concepts. Terminology units can be regarded as units of thinking (in the course of creation of knowledge), units of knowledge (especially for knowledge representation and ordering purposes) and units of communications (if transferred, translated, taught, etc.). Wherever and whenever specialized knowledge or information are created, recorded, communicated, transformed or transferred, terminology plays a crucial role. Terminology units occupy a high share (as a rule – 50-80%) in spoken or written specialized communication. Terminology is indispensable in all information activities.

We are sorry to note that neither public opinion nor government institutions in Latvia have yet come to fully understand the significance of developing national terminology. There is an insufficient understanding of the fact that national terminology is one of the most important pre-requisites for the survival of the Latvian language in multilingual Europe and that the development of computerized information processing systems, as well as competitive scientific and technological products, is fundamentally important in present-day life.

Most activities in the field of Latvian terminology have been left in the hands of a few enthusiasts. In only a few sectors is work being done with the financial and technical support (usually fairly meager support at that) of the Latvian Science Council, various sectoral associations, or individual institutions, companies, funds or universities. Publication of terminological dictionaries is hindered by a lack of resources. Only in a few sectors do terminology specialists have access to sufficiently modern computers and software for the establishment of electronic dictionaries and terminological data bases. The principles for establishing and using terminological work stations have already been determined [20], but application of these principles has been hampered by the lack of resources: the establishment of a single work station which meets present-day standards costs approximately $4,000. Full resolution of these problems would

require at least 20 such work stations under the umbrella of the LATNET Latvian academic network, each work station offering user access to developed terminology, as well as easy access to the information resources of the INTERNET. Despite the fact that in 1995 Latvia became a corresponding member of the ISO and an associated member of the CEN, sufficient efforts are not being made to research the terminological standards of these organizations, to conduct selection and translation processes, or to deal with issues concerning the standardization of Latvian terminology. All of these problems are the direct result of the absence of a unified national policy in the field of information and telecommunications technologies. The fact is that if we fail to resolve these problems in the near term, then we will have great problems in becoming involved in the development of multilingual Europe.

## REFERENCES


1. Galinski, C. and Budin, G. 'Terminology and Standardization in/for Electronic Publishing', Infoterm, 25 - 94 en. (1994).
2. 'Language and Terminology in Europe 2000: Awareness Campaign', seminar, 10-11 November 1994, Riga. Reports, Riga: Institute of Mathematics and Computer Sciences (1994).
3. 'The Unicode Standard: Worldwide Character Encoding, Version 1.0, Vol. 1'. The Unicode Consortium, Addisson-Wesley Publishing Co. (1991).
4. ISO/IEC DIS 10646-1.2 Information Technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and basic multilingual plane.
5. Latvian National Standard LVS 23-93: 'Latviešu tastatūra datoriem' (Latvian keyboard for computers).
6. 'Towards a European Language Infrastructure'. Report by A. Danzin and the Strategic Planning Study Group for the Commission of the European Communities (DG XIII). 31 March 1992. Doc. No. 5210/92.
7. Spektors, A. 'Latviešu valoda un datori' (The Latvian language and computers). Published in the proceedings of the Global Congress of Latvian Sciences, 12-17 July 1991, Rīga, Vol. 5 (1991).
8. Sarkans, U. 'Synthesis of Latvian Linguistic Knowledge', in The First Conference on Baltic Studies in Europe: Linguistics. Rīga (1995), p. 26.
9. Greitāne, I. 'Mašīntulkošanas programma LATRA' (The LATRA automated translation program), LZA Vēstis, intended for publication in 1997.
10. Sampson, G. 'English for the Computer'. The SUSANNE Corpus and Analytic Scheme. Oxford: Clarendon Press (1995).
11. Teubert, W. 'Language Resources: The Foundations of a Pan-European Information Society', in TELRI: Proceedings of the First European Seminar, Tihany, Hungary (1995), pp. 105-128.
12. Erjavec, T., Ide, N., Petkevič, V and Veronis, J. 'MULTEXT-East: Multilingual Text Tools and Corpora for Central and Eastern European Languages', Ibid., pp. 87-97.
13. Sarkans, U. 'Morphemic and Morphological Analysis of the Latvian Language', in Papers in Computational Lexicography COMPLEX '96. Budapest (1995), pp. 219-226.
14. Āboltiņa, I. 'Transcribing of Latvian Names by Computerized Grapheme-to-Phoneme Rules', in Onomastica - Copernicus Research Colloquium. Edinburgh (1996), pp. 47-50.
15. De Sopena, L. 'Speech Recognition: A General Overview', in TELRI, op. cit., pp. 99-103.
16. 'Language Teaching and Learning', in Hearn, P. and Button, D. (eds.). Language Industries Atlas. Amsterdam (1994), pp. 14-15.
17. Carr, R. 'ODA – Applications, Implementations, Conformance, Testing'. - Int. Open Syst. '89: Proc. Conf., London (1989).
18. Skujiņa, V. The Principles of Formation of Latvian Terminology. Rīga: Zinātne (1993). In Latvian.
19. Skujiņa, V. 'Terminology Commission of Latvian Academy of Sciences on the Threshold of its Sixth Decade', in Proceedings of the Latvian Academy of Sciences, Section A. In Latvian (submitted).
20. Borzovs, J. and Čevere, R. 'Lietotāja pamatprasības tipveida darba seansiem ar datorizētu terminoloģisko vārdnīcu'. (Basic user requirements for standard work with a computerized terminological dictionary). - 7th International Baltic Specialists Congress BALTISTICA VII, Rīga (1995).



**Riga Information Technology Institute**
**Skanstes iela 13,**
**LV 1013, Riga,**
**Latvia**
**Tel.: +371 2374764**
**Fax: +371 7821457**
**E-mail: juris.borzovs@dati.lv**

LATVIJAS ZINĀTŅU AKADĒMIJAS VĒSTIS
1990. — Nr. 11 (520). — 124.—128. lpp.

## Zinātne — praksei. Ergonomika

J. Borzovs, A. Kalnroze, J. Rāts

# RAKSTĀMMAŠĪNĀM UN DATORIEM (SKAITĻOTĀJIEM) — ERGONOMISKU LATVIEŠU TASTATŪRU

Rakstāmmašīna ir salīdzinoši jauns cilvēces izgudrojums. Pirmie mēģinājumi radīt rakstāmmašīnu bija 17. gadsimta beigās. 1886. gadā angļu izgudrotājs K. Šoulzs Amerikas Savienotajās Valstīs kopā ar Glidenu un Sulē izgudroja rakstāmmašīnu, ko 1874. gadā sāka ražot firma «Remington». 1898. gadā firma «Underwood» uzsāka ražot rakstāmmašīnas ar t. s. priekšējo piesitienu, kas ļauj kontrolēt rakstīšanas gaitu. No šī brīža rakstāmmašīnas strauji ieviesās praksē.

Lietojot rakstāmmašīnu, iespējams paātrināt darba tempu: ar roku var uzrakstīt 100—120 zīmes minūtē, bet ar rakstāmmašīnu 250—300 un pat vairāk zīmju minūtē jeb 8—10 standartlapu stundā. Bez tam mašīnrakstīšana ceļ darba kultūru un atvieglo turpmāko darbu ar manuskriptu.

Praksē ieviesušās divas mašīnrakstīšanas metodes: t. s. skatīšanās metode un desmitpirkstu jeb aklā metode.

Strādājot pēc skatīšanās metodes, rakstītājs vispirms izlasa teikumu vai tā daļu, tad ar acīm uzmeklē vajadzīgos taustiņus un, izmantojot katras rokas dažus pirkstus, uzraksta izlasīto tekstu. Šis paņēmiens ir mazražīgs un stipri nogurdina rakstītāju, jo uzmanība pārmaiņus jāpievērš gan pārrakstāmajam tekstam, gan rakstāmmašīnas tastatūrai. Lielāka ir arī kļūdīšanās iespēja, grūtāk sasniegt lielu ātrumu.

Lai varētu rakstīt pēc desmitpirkstu metodes, vispirms jāapgūst tastatūra, jāiegaumē katra burta atrašanās vieta. Tomēr šai metodei salīdzinājumā ar iepriekšējo ir vairākas priekšrocības. Rakstot pēc desmitpirkstu metodes, darbā piedalās visi abu roku pirksti. Tādējādi slodze tiek vienmērīgāk sadalīta uz katru pirkstu. Ikviens pirksts apkalpo stingri noteiktu taustiņu zonu un ar laiku tā «iegaumē» taustiņu atrašanās vietas, ka spēj tos uzmeklēt automātiski, bet rakstītājs visu uzmanību var pievērst pārrakstāmajam tekstam [1].

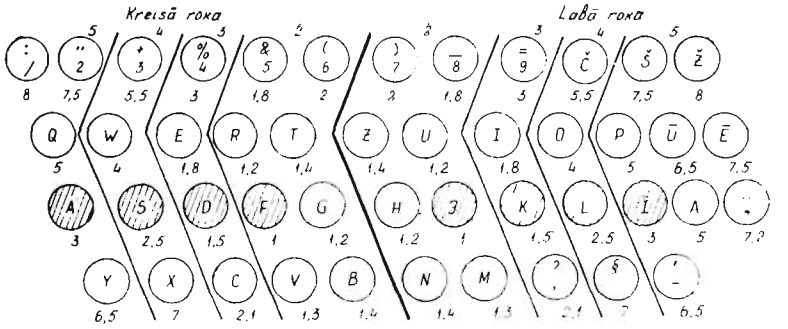
Lai rakstāmmašīna dotu iespēju ražīgi strādāt (sevišķi tas attiecas uz desmitpirkstu metodi), ir jāievēro divi priekšnoteikumi:

— burtu izvietojumam visās tastatūrās jābūt vienādam, t. i., tastatūrai jābūt standartizētai,

— burtu izvietojumam jābūt racionālam, t. i., tam jānodrošina slodzes racionāls sadalījums pa pirkstiem, sabalansēta roku noslodze u. tml., citiem vārdiem — tastatūrai ir jābūt *ergonomiskai.*

Īsi sakot, ir vajadzīgs, lai būtu kārtība un lai šī kārtība būtu saprātīga. Var uzskatīt, ka kārtība pašlaik ir ieviesta, to reglamentē republikas standarts RST 980—85 [2] (sk. 1. un 2. att.). Diemžēl vairāku iemeslu dēļ šo kārtību nevar saukt par saprātīgu.

1. Standarta RST 980—85 noteiktais latviešu alfabēta burtu izvietojums nav racionāls, un tas būtiski samazina darba ražīgumu, strādājot ar desmitpirkstu metodi. Piemēram, latviešu tekstā ļoti reti sastopamie burti «f» un «h», ir novietoti tastatūras centrā un tiek drukāti ar rādītājpirkstiem, turpretī vairāki bieži lietoti burti novietoti tastatūras galos (latviešu valodā visbiežāk lietojamais burts «a» tiek drukāts ar kreisās rokas




*1. att.* Taustiņu iedalījums zonās un grūtības koeficienti. Iesvītroti roku pamatstāvokļa taustiņi

mazo pirkstu, trešais pēc biežuma burts «s» — ar kreisās rokas zeltnesi, devītais pēc biežuma burts «ā» — ar labās rokas mazo pirkstu). Rakstot tekstu latviešu valodā ar rakstāmmašīnu, kas atbilst standartam RST 980—85, ar otro un trešo pirkstu tiek drukāts (vai ievadīts) apmēram 59% teksta, turpretī «gwerty» tastatūrai angļu tekstam šis skaitlis ir apmēram 70% (novērtējumā izmantotie dati par burtu biežumiem latviešu un angļu valodās ņemti no [3]). Burtu izmantošanas biežumi tekstā no 60 000 burtiem (skaitot par burtu arī intervālu starp vārdiem) ir parādīti 2.—4. attēlā zem attiecīgajiem taustiņiem.

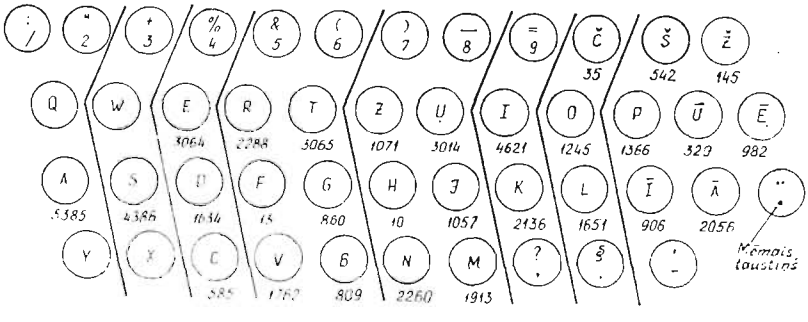
Latviešu rakstāmmašīnām ir raksturīga nepamatoti liela slodze ceturtajiem un piektajiem pirkstiem. Jau sen tika konstatēts [4], ka latviešu tastatūrā «vājākie pirksti (gredzenpirksti un mazie pirksti) tiek pārslogoti, bet spēcīgākie pirksti (rādītājpirksti un viduspirksti) strādā ar daudz mazāku slodzi. Ar to arī izskaidrojams, ka dažas mašīnrakstītājas pēc kursu beigšanas sāk rakstīt pēc neracionālā skatīšanās paņēmiena, jo pārslodzes dēļ tām ātri nogurst un dažreiz pat sāk sāpēt rokas, it īpaši plaukstas gredzenpirksta un mazā pirksta apvidū».

Rezultātā pie mums mašīnrakstīšanas ātrums ir 80—100 burtu minūtē, pretstatā 300—400 burtiem minūtē ārzemēs [4]. Tātad ergonomikas prasībām atbilstošu rakstāmmašīnu ieviešana varētu 3—4 reizes paaugstināt mašīnrakstītāju darba ražīgumu.

2. Saskaņā ar standartu RST 980—85 ciparu «1» drukā kā mazo latīņu alfabēta burtu «l» un ciparu «0» kā lielo latīņu alfabēta burtu «O». Speciālas zīmes šo ciparu drukāšanai standarts neparedz. Šī iemesla dēļ standartu RST 980—85 nevar izmantot datoru tastatūrām.

3. Standarts RST 980—85 nenodrošina iespēju drukāt tekstus latgaliešu valodā, jo tas nesatur burtu «ō».

Ņemot vērā Valodu likumu un ar to saistīto latviešu valodas lietošanas paplašināšanos, racionāla latviešu tastatūras standarta izstrāde pašreiz ir ļoti aktuāla. Turklāt tagad latviešu rakstāmmašīnu vēl ir salīdzinoši nedaudz un jauna standarta ieviešana šodien izmaksās daudz lētāk nekā pēc gada vai diviem.




*2. att.* Pirkstu noslodze tastatūras standartam RST 980—85

process būtu maksimāli efektīvs, ir svarīgi, lai jau pašos pirmajos vingrinājumos apmācāmais varētu ievadīt sakarīgu tekstu. Piedāvājamajā tastatūrā sakarīgu tekstu var ievadīt jau ar četriem taustiņiem (piem., «Atis sita tasi» u. tml.).

Izstrādātais tastatūras standarts ietver 29 no 33 latviešu alfabēta burtiem («ķ», «ļ», «ņ» un «ģ» tiek ievadīti ar speciāla mēmā taustiņa palīdzību), latgaliešu burtu «ō», visus angļu alfabēta burtus, visus ciparus (arī «0» un «1»). Salīdzinot ar pašreizējo tastatūras standartu, piedāvājamajai tastatūrai nav simbolu «%», «/», «&» un paragrāfa zīmes. «Umlauts» ir aizstāts ar apostrofu, kuru var izmantot gan atsevišķi, gan kā mīkstinājuma zīmi mazajam burtam «ģ».

Tabulā dots pašreizējā un piedāvājamā standarta, kā arī vairāku citu piedāvājumu un nestandarta rakstāmmašīnu [4] salīdzinošs vērtējums. Tabulā norādītais vidējais pirkstu nogurums atšķiras no līdzīgas tabulas datiem darbā [6] tāpēc, ka ir koriģēti taustiņu grūtības koeficienti.

Izstrādātā standarta projekta dokumentācija ir iesniegta jauna republikas standarta apstiprināšanai.

## IZMANTOTĀS LITERATŪRAS SARAKSTS


1. *Strebeiko K.* Mašīnrakstīšana. — R.: Liesma, 1978. — 112. lpp.
2. Республиканский стандарт. Машины пишущие. Расположение латышских символов на клавиатуре. РСТ ЛатвССР — 80—85.
3. *Лоренц А., Несауле З.* Статистические закономенрости латышского языка // LPSR ZA Vēstis. — 1966. — Nr. 1. — 65.—72. lpp.
4. *Liepiņš A.* Mašīnrakstīšana. — R., 1928. — 122. lpp.
5. О национальных клавиатурах. — Kirovogradas r/a «Pišmaš» speciālā konstruktoru biroja vēstule LZA Valodas un literatūras institūtam. Nr. 1125/23, 16.09.87.
6. *Borzovs J., Kalnroze A., Rāts J.* Par latviešu rakstāmmašīnas tastatūras standarta projektu. — R.: LVU SC, 1989. — 10 lpp.



Latvijas Universitāte



Iesniegts 11.12.89.

АКАДЕМИЯ НАУК СССР

# ПРОГРАММИРОВАНИЕ

(ОТДЕЛЬНЫЙ ОТТИСК)

2

МОСКВА · 1983

УДК 681.142.2

# АВТОМАТИЧЕСКОЕ ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММ ОБРАБОТКИ ПОСЛЕДОВАТЕЛЬНЫХ ФАЙЛОВ

**Борзов Ю. В.**

Описываются два модельных языка: язык программирования обработки последовательных файлов и язык описания преобразований последовательных файлов. Показывается, что всякое преобразование, записанное на языке программирования, представимо на языке описания.

Отладка программ предполагает выявление и последующее устранение несоответствия между функцией, которую фактически вычисляет программа (так называемая функция программы), и функцией, которую она, по нашему замыслу, должна вычислять. Как традиционное тестирование, так и верификация (доказательство правильности) в случае обнаружения несоответствия названных функций дает весьма скудную информацию о том, что же на самом деле программа вычисляет. Можно предполагать, что автоматическое выявление функции программы по ее тексту будет полезным для отладки программ.

Разработке методов наглядного описания функции программы в последние годы было уделено определенное внимание [1—6]. В некоторых случаях, используя символическое выполнение путей программы [7], удается получить в качестве результата выражение, изображающее ту математическую функцию, которую вычисляет программа [1—3]. Однако, во-первых, полнота представления функции в этом случае сильно зависит от удачи в выборе путей программы для их последующего выполнения и, во-вторых, все подобные построения получены для программ вычислительного характера, написанных, как правило, на Фортране. Работа [5] посвящена задачам преобразования символической информации. В случае обработки данных (Кобол, ПЛ/1 и т. п.) входными данными и результатами работы программы являются файлы. Следовательно, в этих случаях функция программы должна представлять преобразование одних файлов в другие.

В данной статье рассмотрим два языка — язык программирования и язык описания функции программы. Покажем, что все преобразования файлов, которые можно задать на языке программирования, можно автоматически записать на языке описания функции программы. В отличие от текста программы, тоже являющегося записью функции программы, язык описания функции программы оперирует лишь понятиями самих преобразуемых данных — файл, запись и выражения над ними.

## 1. ЯЗЫК ПРОГРАММИРОВАНИЯ

Понятно, что описание функции программы существенно зависит от допустимых структур данных и операций над ними. Поэтому элементы описания, по-видимому, должны различаться для различных языков программирования. Мы сосредоточим внимание на базовом языке [8], являющемся упрощенной абстрактной моделью языков обработки последовательных

файлов. Этот язык характерен также тем, что для него разрешима проблема построения полной системы тестов. Приведем описание базового языка, следуя [8].

Базовый язык. Опишем некоторую абстрактную машину (типа машины Тьюринга), работающую с последовательными файлами. Эта машина называется базовой машиной, а язык программирования, порожденный ею,— базовым языком.

Определим базовую машину (см. рис. 1). Она состоит из управляющего блока, конечного числа односторонних входных лент $A, B, \ldots, C$ и конечного числа односторонних выходных лент $U, V, \ldots, Z$. Управляющий блок содержит конечное число ячеек $a, b, \ldots, v$, называемых внутренними ячейками. В каждую внутреннюю ячейку можно записывать

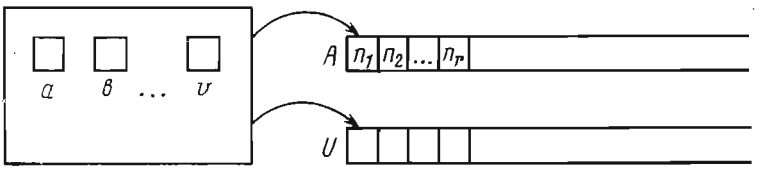



Рис. 1

произвольное целое число. Входные и выходные ленты состоят также из ячеек; в каждую ячейку ленты также может записываться произвольное целое число. Ленты используются для изображения файлов. Под файлом будем понимать переменную, значениями которой являются конечные последовательности целых чисел. Будем говорить, что на ленте записано значение файла $(n_1, n_2, \ldots, n_r)$, если начиная с первой ячейки записаны числа $n_1, n_2, \ldots, n_r$, а остальные ячейки пустые (см. рис. 1). Числа $n_1, n_2, \ldots, n_r$ будем называть записями файла. В дальнейшем, говоря о файле $F$, будем понимать файл, записанный на ленте $F$.

Определим теперь команды, которые может выполнять машина. Пусть $X$ — обозначение произвольной входной ленты, $Y$ — обозначение произвольной выходной ленты, $t$, $v$ — обозначения произвольных внутренних ячеек, $C$ — произвольное целое число (константа).

1) $X \Rightarrow t$ — содержимое обозреваемой ячейки ленты $X$ переписывается во внутреннюю ячейку $t$ (т. е. ячейке $t$ присваивается очередная запись файла $X$), и головка передвигается на одну ячейку вправо. Команда имеет два выхода: выход «+», когда обозреваемая ячейка содержит целое число, и выход «—», когда обозреваемая ячейка пуста. В последнем случае значение ячейки $t$ не меняется.

2) $t \Rightarrow Y$ — содержимое внутренней ячейки $t$ переписывается в обозреваемую ячейку выходной ленты $Y$, а головка записи передвигается на одну ячейку вправо. Команда имеет один выход, который для определенности обозначим через «+».

3) $t \Rightarrow v$ $(C \Rightarrow v)$ — содержимое ячейки $t$ (или константа $C$) переписывается в ячейку $v$. Команда имеет один выход — «+».

4) $t < v$ $(t < C)$ — если содержимое ячейки $t$ меньше содержимого ячейки $v$ (или константы $C$), то управление передается по выходу «+», в противном случае — по выходу «—».

5) СТОП — машина останавливается.

Пример программы. Программу можно наглядно представить граф-схемой. В качестве примера приведем программу слияния двух отсортированных файлов из [8] (см. рис. 2). Предполагается, что все вхождения команд в граф-схему перенумерованы, и выполнение программы всегда начинается с первой команды, а заканчивается командой СТОП.

## 2. ЯЗЫК ОПИСАНИЯ ФУНКЦИИ ПРОГРАММЫ

Сначала введем язык, позволяющий записывать преобразования, проводимые на отдельном пути программы на базовом языке, так называемые функции путей. В общем случае для каждой программы существует неограниченное число функций путей (ФП). Затем будет введен язык формул обработки файлов (ФОФ), позволяющий функцию программы записать конечным числом ФОФ.

1. Язык функций путей. Если $F$ — файл, то через $F_i$ будем обозначать $i$-ю запись файла $F$, где $i \in N$.

Определим *условие*

⟨условие⟩:: = ⌊⟨запись входного файла⟩ ⟨логическая операция⟩ ⟨запись входного файла⟩⌋ | ⌊⟨запись входного файла⟩ ⟨логическая операция⟩ ⟨константа⟩⌋ | ⌊⟨константа⟩ ⟨логическая операция⟩ ⟨запись входного файла⟩⌋

⟨логическая операция⟩:: = $< \mid \geqslant$

⟨константа⟩ ::=0 | ± 1 | ± 2 | ...

Пример. Если $A$ и $B$—входные файлы, то $\lfloor A_{13} < B_{17} \rfloor$ и $\lfloor A_{17} < 1 \rfloor$ суть условия.

Определим *результат*

⟨результат⟩:: = ⟨запись выходного файла⟩ (⟨запись входного файла⟩) | ⟨запись выходного файла⟩ (⟨константа⟩)

Пример. Если $A$ — входной файл, а $U$ — выходной файл, то $U_3$ (17) и $U_1$ ($A_{13}$) суть результаты.

*Условием конца входного файла* назовем букву из алфавита $\{A^*, B^*, \ldots, C^*\}$.

*Описанием преобразования* (ОП) назовем слово, полученное сцеплением записей входных файлов, условий, результатов и условий конца входных файлов по следующим правилам:

1) для каждого $i > 1$, если слово содержит запись входного файла $F_i$, то оно должно содержать в точности по одной записи $F_1, F_2, \ldots, F_{i-1}$, расположенных слева от $F_i$ и не входящих в условия и результаты;

2) справа от условия конца входного файла $F^*$ не должно быть ни одной записи $F_i$, не входящей в условие или в результат;

3) для каждого $i > 1$, если слово содержит запись выходного файла $F_i$, то оно должно содержать в точности по одной записи $F_1, F_2, \ldots, F_{i-1}$, находящихся слева от $F_i$;

4) условие и результат не должны содержать записи входного файла, которые не содержатся в слове слева от них.

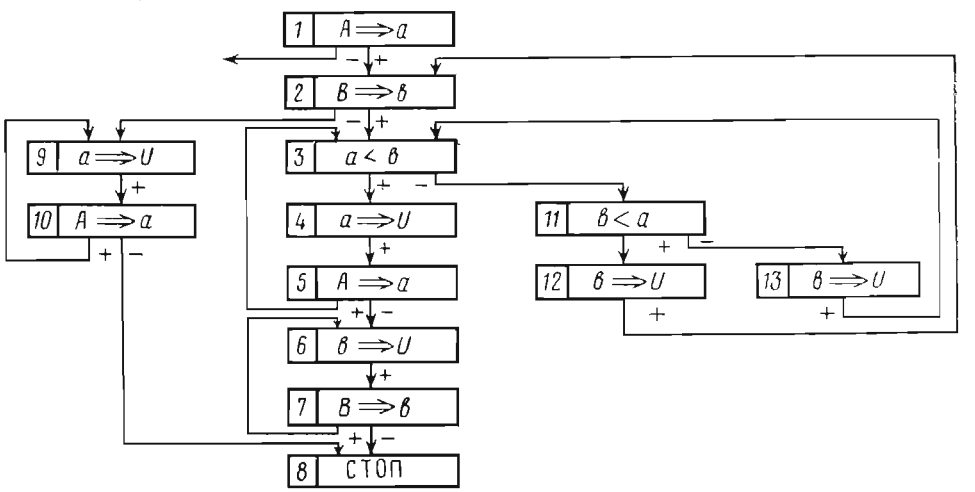



Рис. 2

Пример. Если $A$ и $B$ — входные файлы, а $U$ — выходной файл, то слово $A_1B_1\lfloor A_1 < B_1 \rfloor U_1\,(A_1)\,A^*U_2\,(B_1)\,B^*$ является описанием преобразования (ОП). Кстати, это ОП задает функцию пути (1, 2, 3, 4, 5, 6, 7, 8) программы на рис. 2.

2. Замечания. Содержательно пункты 1) и 3) определения описания преобразования выражают последовательный характер чтения и записи данных, пункт 2) запрещает какое-либо чтение из файла по достижении его конца, а пункт 4) задает очевидное требование: перед обработкой данных их необходимо прочитать.

В примере соответствующие файлам $A$ и $B$ последовательности должны содержать в точности одно число, а соответствующая файлу $U$ последовательность — в точности два числа. При этом число, соответствующее файлу $A$, должно быть меньше числа, соответствующего файлу $B$.

Значением функции пути, определенной данным ОП, на паре последовательности {1} и {2} является последовательность {1, 2}. На паре {1} и {0} эта функция не определена.

3. Язык формул обработки файлов. Обычно программы реализуют такие преобразования, для описания которых требуется неограниченное число ОП. Появляется естественное желание задать все эти преобразования одной или по крайней мере конечным числом формул. Для этого необходимо стянуть «повторяющиеся» части различных ОП в «циклы». Определим *формулу обработки файлов* (ФОФ).

Если $F$ — файл, то обозначим также через $F$ очередную запись файла $F$. Зафиксируем некоторое множество меток $V$.

Определим **Ф-запись**

⟨Ф-запись⟩ ::= ⟨очередная запись входного файла⟩$_v$, где $v \in V$.

Определим **Ф-условие**

⟨Ф-условие⟩ ::= $\lfloor$⟨Ф-запись⟩ ⟨логическая операция⟩ ⟨Ф-запись⟩$\rfloor$ |
$\lfloor$⟨Ф-запись⟩ ⟨логическая операция⟩ ⟨константа⟩$\rfloor$ |
$\lfloor$⟨константа⟩ ⟨логическая операция⟩ ⟨Ф-запись⟩$\rfloor$

Определим **Ф-результат**

⟨Ф-результат⟩ ::= ⟨очередная запись выходного файла⟩ (⟨Ф-запись⟩) |
⟨очередная запись выходного файла⟩ (⟨константа⟩)

Ф-*словом* называется конечное сцепление Ф-записей, Ф-условий и Ф-результатов.

Определим **Ф-цикл**

⟨Ф-цикл⟩ ::= $<$⟨тело цикла⟩$>$

⟨тело цикла⟩ ::= ⟨Ф-слово⟩ | ⟨Ф-цикл⟩ | ⟨тело-цикла⟩ ⟨Ф-слово⟩ |
⟨тело цикла⟩ ⟨Ф-цикл⟩

*Формулой обработки файлов* называется конечное сцепление Ф-слов, Ф-циклов и условий конца входных файлов, если выполняются следующие условия:

а) справа от любого условия конца входного файла $F^*$ нет ни одной Ф-записи $F_v$, не содержащейся в Ф-условии или Ф-результате;

б) Ф-условие и Ф-результат не содержат Ф-запись, которой нет слева от него;

в) условия а) и б) должны выполняться также при удалении из данного сцепления любого количества Ф-циклов.

Пример. Слово $<B_u < A_v\lfloor A_v < B_u\rfloor > A_v\lfloor A_v \geqslant B_u\rfloor U(B_u) > B^*$ является формулой обработки файлов. Содержательно метки $u$ и $v$ обозначают внутренние ячейки, в которых хранятся очередные записи входных файлов. В скобки $<$, $>$ заключаются циклически повторяемые преобразования данных. В частности, $<A_v\lfloor A_v < B_u\rfloor>$ обозначает циклическое чтение записей из файла $A$ во внутреннюю ячейку $v$, которое продолжается, пока прочитанное значение оказывается меньше значения, прочитанного из файла $B$ в ячейку $u$.

4. Процедура генерирования описаний преобразований, исходя из формулы обработки

ф а й л о в. Покажем, что ФОФ объединяет, вообще говоря, неограниченное число различных ОП. Рассмотрим одну из возможных процедур (недетерминированную) генерации множества ОП, исходя из данной ФОФ. Процедура состоит из двух частей — разворачивания и индексирования.

I. а) Если данная ФОФ не содержит Ф-циклов, то переходим к индексированию, если содержит, переходим к б).

б) Находим в ФОФ первый слева Ф-цикл. Изменяем ФОФ, на место этого Ф-цикла записываем его содержимое (т. е. цикл без внешних скобок $<>$), повторенное $k$ раз, где $k$ — любое неотрицательное целое число. Переходим к а).

Описанный процесс разворачивания обязательно закончится, так как глубина вложения в каждом конкретном случае ограничена.

II. а) Проставляем индексы (содержательно — порядковые номера записей) к буквам, обозначающим Ф-записи файлов. В отдельности для каждого входного файла $F$ к первой слева букве $F_v$ проставляем индекс 1, ко второй — 2 и т. д. Ф-записи, находящиеся в Ф-условиях и Ф-результатах, не индексируем и не учитываем при увеличении индексов.

б) Проставляем индексы к буквам, обозначающим очередные записи выходных файлов. Для каждого выходного файла $F$ к первой слева букве проставляем 1, ко второй — 2 и т. д.

в) Проставляем индексы к буквам, обозначающим Ф-записи в Ф-условиях и Ф-результатах. Рассматриваем Ф-условия и Ф-результаты по порядку слева направо. Для каждой Ф-записи $F_v$, находящейся в Ф-условии или Ф-результате, находим ближайшую слева Ф-запись $F_v$ и проставляем к рассматриваемой $F_v$ тот же индекс, который уже проставлен к «левой» $F_v$ согласно пункту а).

г) Удаляем все метки $v \in V$. На этом процедура генерации ОП завершается.

5. П р и м е р  п р и м е н е н и я  п р о ц е д у р ы. Применим вышеописанную процедуру к следующей ФОФ:

$$<B_u<A_v\rfloor A_v<B_u\rfloor>A_v\rfloor A_v>B_u\rfloor U\ (B_u)>B^*$$

Допустим, что первый Ф-цикл при разворачивании повторяется один раз. Тогда получаем

$$B_u<A_v\rfloor A_v<B_u\rfloor>A_v\rfloor A_v>B_u\rfloor U\ (B_u)\ B^*$$

Допустим, что второй Ф-цикл при разворачивании повторяется два раза. Тогда получаем

$$B_uA_v\rfloor A_v < B_u\rfloor A_v\rfloor A_v < B_u\rfloor A_v\rfloor A_v > B_u\rfloor U\ (B_u)\ B^*$$

На этом разворачивание завершено. Переходим к индексированию. После выполнения пунктов а) — г) получаем соответственно:

1. $B_{u,1}A_{v,1}\rfloor A_v<B_{u,}\rfloor A_{v,2}\rfloor A_v<B_{u,}\rfloor A_{v,3}\rfloor A_v> B_u\rfloor U(B_u)\ B^*$
2. $B_{u,1}A_{v,1}\rfloor A_v < B_u\rfloor A_{v,2}\rfloor A_v < B_u\rfloor A_{v,3}\rfloor A_v > B_u\rfloor U_1(B_u)B^*$
3. $B_{u,1}A_{v,1}\rfloor A_{v,1} < B_{u,1}\rfloor A_{v,2}\rfloor A_{v,2} < B_{u,1}\rfloor A_{v,3}\rfloor A_{v,3}>B_{u,1}\rfloor U_1\ (B_{u,1})\ B^*$
4. $B_1A_1\rfloor A_1<B_1\rfloor A_2\rfloor A_2 < B_1\rfloor A_3\rfloor A_3 > B_1\rfloor U_1\ (B_1)\ B^*$

Очевидно, нами получено ОП. Понятно, что, изменяя разворачивание (значения $k$), можно получить неограниченное количество различных ОП. Именно в таком смысле одна ФОФ «объединяет» бесконечно много ОП.

Сразу отметим, что, в силу недетерминированности, язык ФОФ оказался шире базового языка. Существуют преобразования, записанные на языке ФОФ, которые невозможно записать на базовом языке, например, $<<U\ (0)>><U\ (1)>>$ представляет в случае выходного файла $U$ всевозможные последовательности нулей и единиц.

## 3. ОСНОВНАЯ ТЕОРЕМА И АЛГОРИТМ

Т е о р е м а. Существует алгоритм $A$, который для каждой программы П в базовом языке строит конечную систему формул обработки файлов А(П), задающую то же преобразование, что сама программа П.

Полное доказательство теоремы содержится в [6]. Здесь приведем только основные положения доказательства и опишем сам алгоритм. Для построения А (П) сначала необходимо построить дерево реализуемости программы Д(П). Это построение аналогично построению в [8].

1. П о с т р о е н и е д е р е в а р е а л и з у е м о с т и п р о г р а м м ы. Строим развертку программы П в виде дерева: корню дерева (нулевой ярус) приписываем номер первой команды программы, т. е. единицу. Если $q$ — произвольная вершина $k$-го яруса и ей приписан номер команды $n\,(q)$, то из вершины $q$ проводим столько дуг, сколько выходов имеет команда $n\,(q)$. В концах этих дуг строим вершины $(k+1)$-го яруса, которым приписываем номера соответствующих команд. Между вершинами $(k+1)$-го яруса и путями программы устанавливается взаимно-однозначное соответствие: вершине $q_{k+1}$, лежащей на ветви $h = (q_1, q_2, \ldots, q_{k+1})$, сопоставим путь $(n\,(q_1), n\,(q_2), \ldots, n\,(q_{k+1}))$, который обозначим через $\alpha\,(q_{k+1})$.

Далее просматриваем дуги, выходящие из вершин $k$-го яруса, и оборвем (сотрем) дугу $(q_k, q_{k+1})$ вместе с вершиной $(k+1)$-го яруса в следующих случаях:

а) если путь $\alpha\,(q_{k+1})$ реализуем, а путь, полученный из $\alpha\,(q_k)$ добавлением пути $(n\,(q_k), n\,(q_{k+1}))$, нереализуем, т. е. не существует входных данных, на которых программа выполняет данный путь; реализуемость пути устанавливается при помощи процедуры, описанной в [8];

б) если а ) не имеет места, то к вершине $q_k$ приписываем состояние $S\,(\alpha\,(q_k))$.

Состояние — это вся та информация, которая определяет дальнейшее поведение программы. На практике этому понятию соответствует, например, контрольная точка. Формальное определение состояния и метод его построения для базового языка даны в [8]. Правда, для наших целей в состоянии надо сохранять также информацию о том, из которого входного файла получила значение внутренняя ячейка. Далее, просматриваем вершины, лежащие на той же ветви $h$ дерева, на которой находится вершина $q_k$, и обрываем дугу $(q_k, q_{k+1})$, если среди этих вершин имеется такая вершина $p$, что $n\,(p) = n\,(q_k)$ и $S\,(\alpha\,(p)) = S\,(\alpha\,(q_k))$. В этом случае ветвь $h$ будем называть оборванной по повторению, а вершину $p$ будем называть обрывающей для ветви $h$. После обрыва дуг, выходящих из вершин $k$-го яруса, переходим к построению вершин $(k+2)$-го яруса и обрыванию дуг, выходящих из вершин $(k+1)$-го яруса. Если эта процедура завершится (что неизбежно, если программа П имеет конечное число состояний, а в базовом языке дело обстоит именно так [8]), то построенное конечное дерево будем называть деревом реализуемости (обозначим через Д(П)).

Произвольный путь $\alpha = (n_0, n_1, \ldots, n_k)$ *принадлежит* ветви $h = (q_1, q_2, \ldots, q_i, \ldots, q_{i+k}, \ldots, q_l)$, начиная с вершины $q_i$ по вершину $q_{i+k}$, если $n_0 = n\,(q_i)$, $n_1 = n\,(q_{i+1})$, $\ldots$, $n_k = n\,(q_{i+k})$. Если $q_{i+k}$ — последняя вершина ветви $h$, то будем говорить, что путь принадлежит концу ветви $h$.

Произвольный путь $\alpha$ *согласуется* с деревом реализуемости Д(П), если выполняется одно из двух следующих утверждений:

1. $\alpha$ принадлежит ветви Д(П), заканчивающейся вершиной, которая соответствует команде СТОП;

2. $\alpha$ можно разделить на такие части $\alpha_1, \alpha_2, \ldots, \alpha_n$ $(\alpha = \alpha_1 + \alpha_2 + \ldots + \alpha_n)$, что а) любой из путей $\alpha_i$ $(i = 1, 2, \ldots, n-1)$ принадлежит концу некоторой оборванной по повторению ветви $h_k$; б) любой из путей

$\alpha_i$ ($i = 2, 3, \ldots, n$) принадлежит $h_k$, начиная с обрывающей вершины ветви $h_{k-1}$. Знак «+» обозначает операцию сцепления путей.

Лемма 1 [9]. Пусть описанный выше алгоритм строит для программы П конечное дерево реализуемости Д (П). Тогда произвольный путь программы реализуем тогда и только тогда, когда этот путь согласуется с Д(П).

2. Построение ФОФ из Д(П).

Лемма 2 [6]. Для каждой программы $П_1$ в базовом языке существует функционально эквивалентная ей программа $П_2$, не содержащая команд типа $t \Rightarrow v$.

Из леммы следует, что без ограничения общности можем считать, что данная нам программа П не содержит команд пересылки значений из одной внутренней ячейки в другую.

Находим все ветви Д(П), которые начинаются с вершины нулевого яруса и заканчиваются вершиной, соответствующей команде СТОП (СТОП-ветви). Если в Д(П) нет ни одной такой ветви, то ФОФ — пустая (ни на каких входных данных программа не завершает работу нормально). Для каждой СТОП-ветви находим соответствующий ей путь. Просматриваем команды пути по порядку до команды, соответствующей обрывающей вершине.

Если рассматриваемой командой является

1) $X \Rightarrow t$ и выход из нее по «+», пишем $X_t$;
$X \Rightarrow t$ и выход из нее по «—», пишем $X^*$;

2) $t \Rightarrow Y$, то пишем $Y$ ($X_t$), где $X$ — файл, чья запись находится в $t$ (если в $t$ содержится константа $C$, то пишем $Y$ ($C$));

3) $C \Rightarrow t$, то не делаем ничего;

4) $t < v$ ($t < C$) и выход по «+», то пишем $\lfloor X_t < Z_v \rfloor$ ($\lfloor X_t < C \_$), где $X$ и $Z$ — файлы, чьи записи находятся в $t$ и $v$, или константы, содержащиеся в $t$ и $v$;

$t < v$ ($t < C$) и выход по «—», то пишем $\lfloor X_t \geqslant Z_v \rfloor$ ($\lfloor X_t \geqslant C \_$);

если $t$ и $v$ обе (или $t$ в случае команды $t < C$) содержат константы, то ничего не делается;

5) СТОП, то заканчиваем построение ФОФ для данной СТОП-ветви и переходим к обработке следующей СТОП-ветви.

Информация для проведения действий 1) — 5) содержится в состояниях, приписанных к вершинам Д(П) во время его построения.

Если при выполнении описанных действий рассматриваемой команде будет соответствовать обрывающая вершина, то выясняем, сколько оборванных вершин имеется в Д(П) для этой обрывающей вершины.

I. Если оборванная вершина одна, то пишем $<$, находим путь, ведущий от обрывающей вершины к оборванной, и обрабатываем этот путь согласно 1) — 5), пока не доходим до оборванной вершины; эту последнюю не обрабатываем, а пишем $>$.

II. Затем продолжаем обработку пути, соответствующего СТОП-ветви, начиная с команды, которая соответствует обрывающей вершине, но на этот раз считаем ее «обычной» вершиной.

III. Если оборванных вершин несколько, то сначала пишем $<$. Затем обрабатываем каждый путь, ведущий из обрывающей в оборванные вершины согласно I. По окончании пишем $>$. Переходим к II.

Если при обработке некоторой обрывающей вершины (I—III) попадаем на следующую обрывающую вершину, то прерываем обработку предыдущей обрывающей вершины, запоминая ситуацию и начиная обработку следующей обрывающей вершины. Когда эта последняя обработана, продолжаем обработку предыдущей.

Можно убедиться, что описанная процедура строит конечную систему ФОФ.

Завершение доказательства теоремы.

Лемма 3 [6]. Существует алгоритм $A_1$, который для каждого согла-

сующегося с Д (П) СТОП-пути $\alpha$ строит описание преобразования $A_1(\alpha)$, задающее функцию пути $\alpha$.

Обозначим процедуру генерации конкретного ОП из ФОФ через Г. Каждый конкретный вариант разворачивания и индексирования в процедуре Г назовем *применением* процедуры Г и обозначим через Г′.

*СТОП-путь*: это путь, начинающийся первой командой и завершающийся командой СТОП.

Лемма 4 [6]. Для каждого согласующегося с Д(П) СТОП-пути $\alpha$ существует такое применение процедуры Г к системе ФОФ А(П), что $\Gamma'(A(\Pi))=A_1(\alpha)$.

Лемма 5 [6]. Для каждого применения процедуры Г к системе ФОФ А(П) существует согласующийся с Д(П) СТОП-путь $\alpha$, что $\Gamma'(A(\Pi))=A_1(\alpha)$.

Теорема доказана.

## 4. ПРИМЕР ФОРМУЛ ОБРАБОТКИ ФАЙЛОВ

В качестве примера приведем систему ФОФ, построенную для программы из [8] (см. рис. 2) вышеописанным алгоритмом.

1) $A_aB_b\lessdot\lessdot\lfloor A_a<B_b\rfloor U(A_a)A_a\gtrdot\lessdot\lfloor A_b\geqslant B_b\rfloor\lfloor B_b<A_a\rfloor U(B_b)B_b\gtrdot\gtrdot$
$\lfloor A_a<B_b\rfloor U(A_a)A^*U(B_b)B_b\lessdot U(B_b)B_b\gtrdot U(B_b)B^*$
2) $A_aB_b\lessdot\lessdot\lfloor A_a<B_b\rfloor U(A_a)A_a\gtrdot\lessdot\lfloor A_a\geqslant B_b\rfloor\lfloor B_b<A_a|\,U(B_b)B_b\gtrdot\gtrdot$
$\lfloor A_a<B_b\rfloor U(A_a)A^*U(B_b)B^*$
3) $A_aB_b\lessdot\lessdot\lfloor A_a<B_b\rfloor U(A_a)A_a\gtrdot\lessdot\lfloor A_a\geqslant B_b\rfloor\lfloor B_b<A_a\rfloor U(B_b)B_b\gtrdot\gtrdot$
$\lfloor A_a\geqslant B_b\rfloor\lfloor B_b<A_a\rfloor U(B_b)B^*U(A_a)A_a\lessdot U(A_a)A_a\gtrdot U(A_a)A^*$
4) $A_aB_b\lessdot\lessdot\lfloor A_a<B_b\rfloor U(A_a)A_a\gtrdot\lessdot\lfloor A_a\geqslant B_b\rfloor\lfloor B_b<A_a\rfloor U(B_b)B_b\gtrdot\gtrdot$
$\lfloor A_a\geqslant B_b\rfloor\lfloor B_b<A_a\rfloor U(B_b)B^*U(A_a)A^*$
5) $A_aB^*\lessdot U(A)_aA_a\gtrdot U(A_a)A^*$

Данная система ФОФ не является минимальной. Метки также являются излишними, так как записи файла $A$ всегда считываются в ячейку $a$, а записи файла $B$ — в ячейку $b$. После очевидных упрощений получаем

1) $AB\lessdot\lessdot\lfloor A<B\rfloor U(A)A\gtrdot\lessdot\lfloor B<A\rfloor U(B)B\gtrdot\gtrdot\lfloor A<B\rfloor U(A)A^*$
$\lessdot U(B)B\gtrdot U(B)B^*$
2) $AB\lessdot\lessdot\lfloor A<B\rfloor U(A)A\gtrdot\lessdot\lfloor B<A\rfloor U(B)B\gtrdot\gtrdot\lfloor B<A\rfloor U(B)B^*$
$\lessdot U(A)A\gtrdot U(A)A^*$
3) $AB^*\lessdot U(A)A\gtrdot U(A)A^*$

Рассматривая данную систему формул, можно убедиться, что не предполагается равенство двух записей из файлов $A$ и $B$, а также отсутствие записей в файле $A$. Это свидетельствует об ошибках в программе.

* * *

Понятно, что вышеописанные ФОФ и процедура генерации конкретных ОП и ФОФ применимы только для весьма узкого класса преобразований данных — это преобразования последовательных файлов, не включающие сложные преобразования данных полей: арифметику, сцепление и т. п. Практическое применение ФОФ в отладке программ будет зависеть от расширений, необходимых для отражения структур и преобразований данных реальных языков программирования, и в первую очередь от наглядности и обозримости получаемых формул.

## ЛИТЕРАТУРА


1. *King J. C.* A new approach to program testing.— Proc. Intern. Conf. on Reliable Software, 1975, p. 228.
2. *Boyer R. S., Elspas B., Levitt K. N.* SELECT — a formal system for testing and debugging programs by symbolic execution.— Proc. Intern. Conf. on Reliable Software, 1975, p. 234.

3. *Howden W. E.* Symbolic testing and the DISSECT symbolic evaluation system.— IEEE Trans. Software Eng., 1975, SE—3, № 4, p. 266.
4. *Бичевский Я. Я., Борзов Ю. В.* К вопросу о формальном описании функции, реализуемой программой.— Тезисы докл. Первой междунар. конф. молодых ученых «Проблемы проектирования и применения дискретных систем в управлении». Минск, 1977, с. 207.
5. *Ефимкин К. Н., Задыхайло И. Б.* О верификации программ на одном языке.— Программирование, 1980, № 2, с. 69.
6. *Борзов Ю. В.* Автоматическое построение описания функции, реализуемой программой ЭВМ.— Рукопись деп. в ВИНИТИ 25 февраля 1981 г., № 887—81 Деп.
7. *Борзов Ю. В.* Тестирование программ с использованием символического выполнения.— Программирование, 1980, № 1, с. 51.
8. *Барздинь Я. М., Бичевский Я. Я., Калниньш А. А.* Построение полной системы примеров для проверки корректности программ.— В кн.: Уч. зап. Латв. гос. ун-та. Рига, 1974, т. 210, с. 152.
9. *Бичевский Я. Я.* Автоматическое построение полных систем примеров: Дис. на соискание уч. ст. канд. физ.-мат. наук. Рига, 1977.

Поступила в редакцию 9.X.1981

УДК 681.3.06

# О РЕАЛИЗАЦИИ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ ПАКЕТА ПРОГРАММ РАЗМЕЩЕНИЕ

**Ещенко В. Г.**

В статье рассматриваются некоторые особенности технологии построения разработанного в ИПМаш АН УССР пакета программ РАЗМЕЩЕНИЕ для решения задач рационального размещения геометрических объектов, позволившие реализовать пакет на языке высокого уровня.

Пакет программ (ПП) РАЗМЕЩЕНИЕ [1] предназначен для решения класса оптимизационных задач нерегулярного размещения геометрических объектов в заданных областях. Задачи этого класса являются математическими моделями таких важных практических задач, как построение рациональных планов раскроя промышленных материалов, проектирование схем генеральных планов промышленных предприятий, размещение оборудования крупных цехов и производств, компоновка элементов радиоэлектронной аппаратуры и т. д.

*Анализ и синтез задач.* В результате анализа содержательных и формальных постановок задач выделены следующие их характерные особенности [2]. Каждая задача класса включает четыре компонента: размещаемые объекты, область размещения, ограничения и критерий качества размещения. Компонент «размещаемые объекты» характеризует геометрическую форму объектов и включает следующие элементы: прямоугольники, круги, выпуклые и невыпуклые многоугольники. Компонент «область размещения» характеризует форму и топологию области размещения и включает ограниченные и неограниченные прямоугольные области, круги, несвязные и многосвязные многоугольные области. Элементы компонента отражают различные технологические требования и ограничения и включают такие понятия, как выполнение минимальных расстояний между объектами, постоянство или изменение в заданных пределах ориентации объектов в процессе размещения, наличие в области размещения зон запрета, требование сквозных резов и т. д. Элементы компонента «критерий качества размещения» задают преследуемую при размещении объектов цель: минимум длины занятой части области размещения, минимум длины сети, связывающей объекты, максимум коэффициента заполнения, минимум отклонения центра тяжести системы размещенных объектов от заданной точки и т. д.

*Методы решения.* Задачи рассматриваемого класса являются задачами математического программирования и обладают рядом характерных особенностей [3, 4], которые обусловили двухэтапный процесс поиска оптимального размещения, включающий построение вариантов размещения и их целенаправленный перебор. Для поиска оптимального размещения разработаны специальные методы оптимизации: метод значимых переменных [4] и метод сужающихся окрестностей [5].

Latvijas PSR Tautas izglītības ministrija
Latvijas Valsts universitātes Skaitļošanas centrs

J. Borzovs, A. Kalnroze, J. Rāts

# Par latviešu rakstāmmašīnas ergonomiskās tastatūras standarta projektu

RĪGA 1989

Latvijas PSR Tautas izglitibas ministrija
Latvijas Valsts universitates Skaitļošanas centrs

J. Borzovs, A. Kalnroze, J. Rāts

# Par latviešu rakstāmmašīnas ergonomiskās tastatūras standarta projektu

## Pielikums

UDK 681.3

Par latviešu rakstāmmašīnas ergonomiskās tastatūras standarta projektu. J. Borzovs u.c. - Rīga: LVU SC, 1989.- 10 lpp.

Autori piedāvā ergonomikas prasībām atbilstošu latviešu rakstāmmašīnas tastatūras projektu, kurš garantē augstražīgu darbu, strādājot ar desmitpirkstu (aklo) metodi.

4 zīmējumi, bibliogrāfija - 4 nosaukumi.

Latviešu valodas tiesību atjaunošana ir saistīta ari ar tādiem prozaiskiem pasākumiem, kā uzņēmumu, organizāciju un iestāžu apgāde ar latviešu rakstāmmašīnām un datoriem (skaitļotājiem) ar latviešu tastatūrām. Pirms šī apgāde ir ieguvusi masveida raksturu, ir jānovērtē latviešu pašreizējā rakstāmmašīnas tastatūra un jāizlemj, vai tajā nebūtu lietderīgi izdarīt izmaiņas.

Praksē tiek lietotas divas mašīnrakstīšanas metodes - skatīšanās metode un desmitpirkstu metode [1]. Pirmo izmanto neprofesionāļi, otro pārsvarā lieto profesionālas mašīnrakstītājas. Desmitpirkstu metodes pamatā ir taustiņu sadalījums fiksētās zonās tādējādi, ka katrs pirksts apkalpo noteiktu taustiņu zonu (1. zīm.). Strādājot ar desmitpirkstu metodi, taustiņu novietojums tastatūrā būtiski ietekmē mašīnrakstītājas darba efektivitāti. Nepareizs taustiņu izvietojums tastatūrā samazina mašīnrakstītāju darba ražīgumu un līdz ar to rada ekonomiskus zaudējumus.

Latviešu rakstāmmašīna, tāpat kā virkne citu uz latīņu alfabēta bāzētu valodu rakstāmmašīnu, ir veidota pēc angļu standarta tastatūras "qwerty" parauga. Trūkstošos burtus vienkārši pievienoja tastatūras galos (burtu "ķ", "ļ", "ņ" un "ģ" ievadam izveidoja speciālu "zēmo" taustiņu mikstinājuma zīmei), tādējādi padarot latviešu teksta drukāšanu iespējamu. Faktiskais stāvoklis ir fiksēts Latvijas republikāniskajā standartā RST 980-85 [2], kurš reglamentē latviešu rakstāmmašīnu ražošanu pašreiz (sk. 1. un 2. zīm.) un kuram ir sekojoši trūkumi:

1. Standarts RST 980-85 nosaka neracionālu latviešu alfabēta burtu izvietojumu, kas būtiski samazina darbā ražīgumu, strādājot ar desmitpirkstu (aklo) metodi. Latviešu tekstā ļoti reti sastopamie burti "f" un "h", piemēram, ir novietoti tastatūras centrā un tiek drukāti ar rādītājpirkstiem. Vairāki bieži lietoti burti, turpretī, novietoti tastatūras galos (visbiežākais latviešu valodas burts "a" tiek drukāts ar kreisās rokas mazo pirkstu, trešais pēc biežuma burts "s" - ar kreisās rokas zeltnesi, devītais pēc biežuma burts "ā"- ar labās rokas mazo pirkstu). Rakstot latviešu valodas tekstu ar rakstāmmašīnu, atbilstošu

standartam RST 980-88, ar otro un trešo pirkstu tiek ievadīts apmēram 50% teksta, turpretī "qwerty" tastatūrai angļu tekstam šis skaitlis ir apmēram 70% (novērtējumā izmantotie dati par burtu biežumiem latviešu un angļu valodās ņemti no [3]). 2. zīmējumā ir parādīti burtu taustiņu izmantošanas biežumi tekstā no 60000 burtiem (skaitot par burtu ietervālu starp vārdiem).

Latviešu rakstāmmašīnām ir raksturīga nepamatoti liela slodze ceturtajiem un piektajiem pirkstiem. Jau Latvijas Republikas laikā tika konstatēts [4], ka latviešu tastatūrā "vājākie pirksti" (gredzenpirksti un mazie pirksti) tiek pārslogoti, bet spēcīgākie pirksti (rādītājpirksti un viduspirksti) strādā ar daudz mazāku slodzi. Ar to arī izskaidrojams, ka dažas mašīnrakstītājas pēc kursu beigšanas sāk rakstīt pēc neracionālā skatīšanās papīrmiena, jo pārslodzes dēļ tām ātri nogurst un dažreiz pat sāk sāpēt rokas, it īpaši plaukstas gredzenpirksta un mazā pirksta apvidū". Rezultātā pie mums vidējais mašīnrakstīšanas ātrums ir 80-100 burtu minūtē, pretstatā 300-400 burtiem minūtē ārzemēs [4]. Tātad, ergonomikas prasībām atbilstošu rakstāmmašīnu ieviešana varētu 3-4 reizes paaugstināt mašīnrakstītāju darba ražīgumu.

2. Saskaņā ar standartu RST 980-88 ciparu "1" drukā kā mazo latiņu alfabēta burtu "l" un ciparu "0" - kā lielo latiņu alfabēta burtu "O". Speciālas zīmes šo ciparu drukāšanai standarts neparedz. Šī iemesla dēļ standartu RST 980-88 nevar izmantot datoru tastatūrām.

3. Standarts RST 980-88 nenodrošina iespēju drukāt tekstus latgaliešu valodā, jo tas nesatur burtu "ō".

Veidojot un vērtējot latviešu rakstāmmašīnas ergonomisko tastatūru, tika ņemti vērā sekojoši kritēriji.

*Pirkstu noslodze.* Rādītāja un vidus pirksti ir stiprāki un veikļāki par zeltnesi un mazo pirkstu. Tāpēc burti uz tastatūras ir jāizvieto tā, lai pēc iespējas lielāku teksta daļu varētu ievadīt ar rādītāja un vidus pirkstiem.

*Pirkstu izkustināšana no pamatstāvokļa.* Strādājot ar desmitpirkstu paņēmienu, rokas pastāvīgi atrodas noteiktā vietā uz tastatūras. Šo roku stāvokli sauc par pamatstāvokli. Līdz ar to aktīvas zīmes var ievadīt, neizkustinot rokas no pamatstāvokļa (ar

abu roku 2-5 pirkstiem). Lai ievaditu citas zīmes, attiecīgo pirkstu atvirza no pamatstāvokļa, nospiež vajadzīgo taustiņu un pēc tam atgriež pirkstu pamatstāvokli. Tāpēc zīmju ievads ar taustiņiem, kuri neatrodas pirkstu pamatstāvokļa vietās, prasa papildus laiku un slodzi. Šī papildus slodze un laiks ir dažādi dažādiem taustiņiem. Ņemot vērā šo faktoru, kā arī to, ka pirksti atšķiras spēciguma un veikluma ziņā, katram taustiņam tika dots novērtējums - taustiņa grūtības koeficients. Nosakot taustiņu grūtības koeficientus, tika aptaujātas profesionālas mašīnrakstītājas.

Grūtības koeficients ā raksturo, cik reižu "grūtāk" simbolu ievadīt ar doto taustiņu, salīdzinājumā ar taustiņu, atbilstošu rādītājpirksta pamatstāvoklim. Taustiņu grūtības koeficienti ir attēloti 1. zīmējumā zem attiecīgajiem taustiņiem.

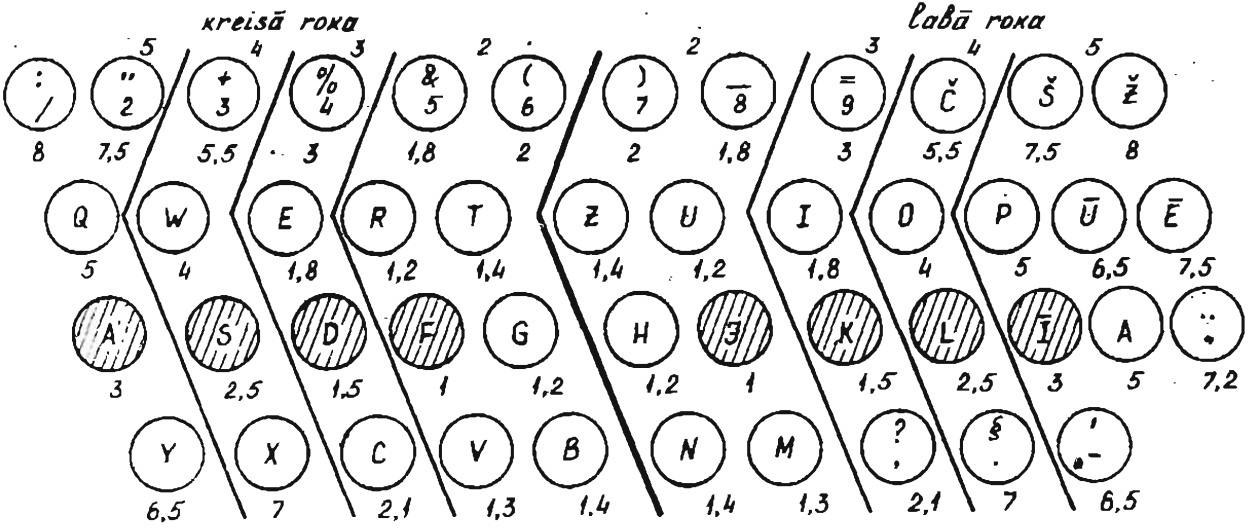
*Ritms.* Ir konstatēts (sk. piem. [1]), ka mašīnrakstītāja mazāk nogurst un ātrāk strādā tad, ja tekstā vairāk ir tādu vienu otram sekojošu burtu pāru, kurus ievadot ir jāmaina rokas. Varam uzskatīt, ka burta ievadīšanas grūtība ir atkarīga no tā, vai iepriekšējais burts tika ievadīts ar to pašu vai ar otru roku. Uzskatām, ka grūtība bez tam mainās atkarībā no tā, vai iepriekšējā zīme ievadīta ar to pašu pirkstu, vai ar citu tās pašas rokas pirkstu.

3. zīmējumā parādīts piedāvājamais latviešu rakstāmmašīnas ergonomiskās tastatūras projekts, zem taustiņiem zīmējumā ir parādīts to izmantošanas biežums. Tastatūra ietver 29 no 33 latviešu alfabēta burtiem ("ķ", "ļ", "ņ" un "ģ" tiek ievaditi ar speciāla mēmā taustiņa palīdzību), latgaliešu burtu "ō", visus angļu alfabēta burtus, ciparus "0" un "1". Salīdzinot ar pašreizējo tastatūras standartu, piedāvājamajai tastatūrai nav simbolu "%", "/", "&" un paragrāfa zīmes. "Umlauts" ir aizstāts ar apostrofu, kuru var izmantot gan atsevišķi, gan kā mikstinājuma zīmi mazajam burtam "ģ".

4. zīmējumā dots pašreizējā un piedāvājamā standartu, kā arī dažu nestandarta rakstāmmašīnu [4] salīdzinošs vērtējums.

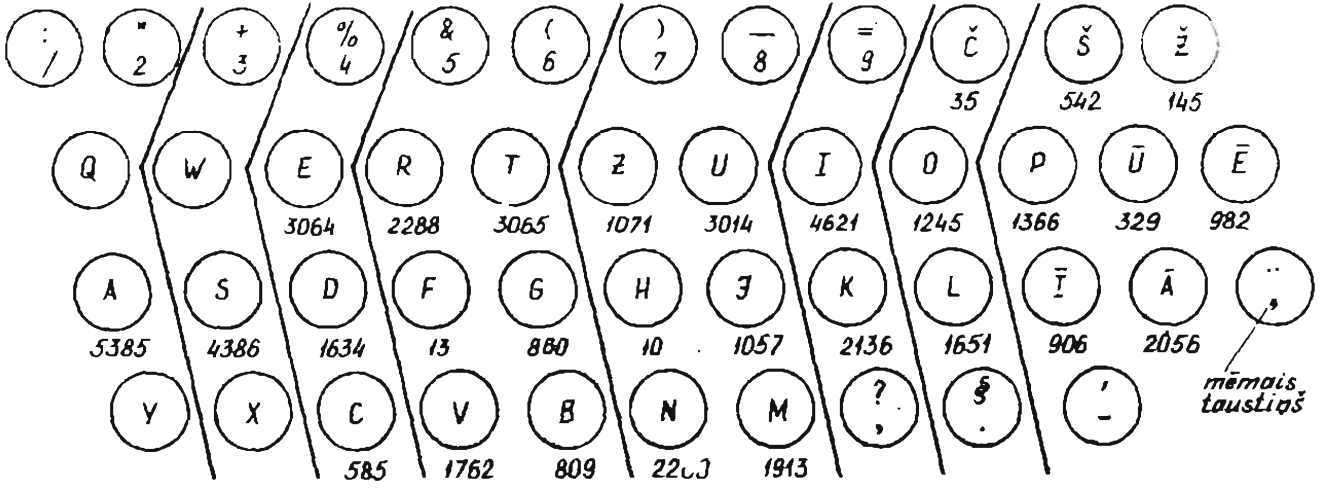
**Literatūra**

1. K. Strebeiko. Mašīnrakstīšana. - Rīga: Liesma, 1978. - 112 lpp.
2. Республиканский стандарт. Машины пишущие. Расположение латышских символов на клавиатуре. РСТ Латв. ССР 980-85.
3. А. Лоренц, З. Несауле. Статистические закономерности латышского языка/ Latvijas PSR ZA vēstis. - N 1(222). - 1966. - Lpp. 65-72.
4. A. Liepiņš, Mašīnrakstīšana. - Rīga, 1928. - 122 lpp.

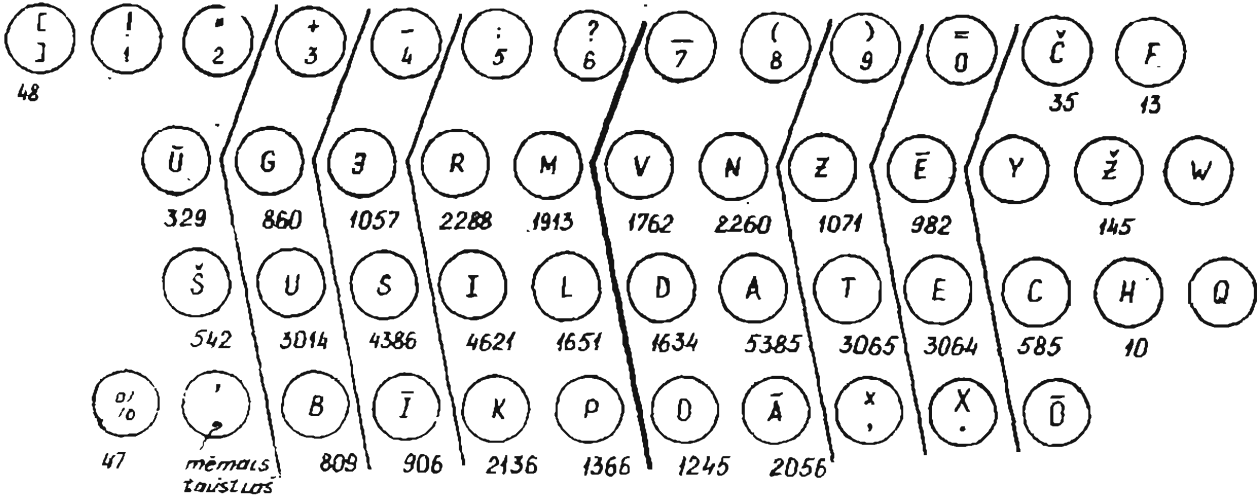



1. zīmējums. Taustiņu iedalījums zonās, taustiņu grūtības koeficienti.

▨ - roku pamatstāvokļa taustiņi




2. zīmējums. Pirkstu noslodze tastatūras standartam RST 980-85




3. zīmējums. Latviešu rakstāmmašīnas ergonomiskās tastatūras projekts. Pirkstu noslodze.

| Tastatūra | Ievadīts ar 2. un 3. pirkstu (%) | Ievadīts ar 8 pamattaustiņiem (%) | Zīmju pāris ievadīts maiņot rokas (%) | Vidējais pirkstu nogurums |
|---|---|---|---|---|
| Underwood 2 | 59.12 | 31.91 | 52.36 | 3.85 |
| Kontinental | 59.12 | 34.03 | 51.64 | 2.50 |
| RST 980-85 | 59.12 | 34.03 | 53.82 | 2.33 |
| Piedāvājamais projekts | 78.16 | 51.56 | 59.57 | 1.83 |

$$\text{Vidējais pirkstu nogurums} = \frac{\text{nospiesto taustiņu grūtības koeficientu summa}}{\text{burtu skaits virknē}}$$

4 zīmējums. Tastatūru salīdzinošs vērtējums.

Viss par datoru

# PERSONĀLIE DATORI

ANGĻU
LATVIEŠU
KRIEVU

SKAIDROJOŠĀ VĀRDNĪCA

# Personālie datori

## Angļu—latviešu—krievu skaidrojošā vārdnīca



1998
Izdevējs a/s DATI

*Šī vārdnīca izdota ar Latvijas Izglītības fonda mērķu programmas "Izglītībai, zinātnei un kultūrai" un a/s "Dati" atbalstu*

## Vārdnīcas sastādītāji

Izmantojot Latvijas Zinātnes padomes finansiālo atbalstu vārdnīcā iekļauto terminoloģiju izstrādājuši un vārdnīcu sastādījuši LU Elektronikas un datorzinātņu institūta darbinieki un pieaicināto speciālistu grupa šādā sastāvā:

Dr. h. dat. A. Baums, Dr. dat. J. Borzovs, Dr. dat. A. Gobzemis, prof. I. Freibergs, Dr. dat. G. Fricnovičs (darba vadītājs), Dr. dat. I. Ilziņa. Redaktore — Dr. h. fil., prof. V. Skujiņa.

Vārdnīcā iekļautos latviešu terminus apspriedusi un rekomendējusi izmantošanai LZA Terminoloģijas komisijas informātikas apakškomisija.

## Vārdnīcas lietotājiem

Ceļā pie lasītājiem dodas otra datorzinātnes terminu skaidrojošā vārdnīca. Pirmā šīs jomas latviešu terminoloģijai veltītā angļu-krievu-latviešu skaidrojošā vārdnīca "Datu pārraides un apstrādes sistēmas" (izdevējs: a/s "SWH Informatīvās sistēmas") publicēta 1995. gadā. Sastādot šo vārdnīcu, galvenā uzmanība tika pievērsta vienotas latviešu terminu sistēmas izveidei, kas aptvertu datu apstrādes un pārraides sistēmu uzbūves, darbības un lietojumu pamatterminus. Plašā personālo datoru izmantošana praktiski visās zinātnes un tautsaimniecības nozarēs, kā arī dažādu sadzīves rakstura problēmu risināšanā rosināja sastādītājus detalizētāk pievērsties tieši ar šiem datoriem saistītās latviešu terminoloģijas veidošanai un sakārtošanai. Tāpēc, strādājot pie jaunās vārdnīcas, tās sastādītāji par galveno uzdevumu izvirzīja ar personālo datoru arhitektūru, aparatūru, programmatūru un lietojumiem saistītās latviešu terminoloģijas izstrādāšanu. Abu vārdnīcu tematika ir visai cieši saistīta un ir uzskatāma par turpinājumu vienotas latviešu terminu sistēmas izstrādāšanai datorzinātnē. Lai gūtu pietiekami pilnīgu priekšstatu par latviešu terminoloģijas veidošanos datorzinātnēs, būtu vēlams iepazīties un izmantot abas šīs vārdnīcas.

Sastādītāji cer, ka jaunā vārdnīca būs noderīgs palīgs gan visai plašajai personālo datoru lietotāju saimei, gan speciālās literatūras tulkotājiem, gan arī zinātniskajiem darbiniekiem, augstskolu mācību spēkiem un studentiem, kā arī informātikas skolotājiem un skolēniem, kas nopietni grib apgūt iemaņas darbam ar personālajiem datoriem.

Šķirkļi vārdnīcā sastāv no angļu termina, tam atbilstošā latviešu termina (terminiem) un tā ekvivalenta krievu valodā, kā arī no termina skaidrojuma latviešu valodā un šķirkļa numura. Šķirkļi sakārtoti angļu alfabēta secībā. Lai atvieglotu lasītājam darbu ar vārdnīcu, tai pievienoti latviešu un krievu terminu alfabētiskie rādītāji, kuros norādīti terminiem atbilstošie šķirkļu numuri. Terminu skaidrojumiem vārdnīcā ir pakārtota nozīme. Skaidrojumu galvenais uzdevums ir sniegt lasītājam vispārīgu priekšstatu par termina saturu, tā lietojuma robežām un terminu savstarpējām attiecībām. Lai atklātu terminu savstarpējās saites un dotu lasītājam iespējami pilnīgāku priekšstatu

par vārdnīcā iekļautajiem terminiem, tāpat kā 1995. gadā publicētajā vārdnīcā, izmantotas trīs veidu šķērsatsauces. Pirmais jebkura vārdnīcā iekļautā termina lietojums kāda cita termina skaidrojumā ir pasvītrots. Atsauce "Sk." norāda, ka aplūkojamais termins ir kāda cita termina sinonīms vai akronīms un lasītājam jāiepazīstas ar atsaucē minētā termina skaidrojumu. Atsauce "Sk. arī" norāda, ka tajā minētais termins (termini) ir saistīti ar aplūkojamo terminu un, iepazīstoties ar tā (to) skaidrojumiem, var iegūt papildinformāciju par aplūkojamo terminu.

## Pateicības

Tā kā vārdnīcā iekļautās terminoloģijas izstrādāšanas laikā, izmantojot tīklu *Internet*, ar darba grupas priekšlikumiem bija iespējams iepazīties visai plašam interesentu lokam, tad gan no Latvijas, gan arī no ASV, Austrālijas, Kanādas un citām valstīm, kur dzīvo un strādā latvieši, vārdnīcas sastādītājiem tika sūtītas gan emocionāla, gan konstruktīva rakstura piezīmes un priekšlikumi. Mēs sirsnīgi pateicamies visiem, kas ar kādu rosinošu domu mums palīdzēja izvairīties no kļūdām un meklēt iespējami labākus latviešu terminu variantus, kā arī Rīgas Informācijas tehnoloģijas institūta darbiniecēm Dr. dat. R. Čeverei, Maģ. dat. M. Lučkinai un Bak. dat. S. Bērziņai, kuras nodrošināja šādas informācijas apmaiņas iespējas.

Vārdnīcas sastādītāji izsaka sirsnīgu pateicību A. Ilgunas un D. Šepeļevai par to pacietību un rūpību, ar kādu tika veikta vārdnīcas teksta datorizēta apstrāde.

Vārdnīcas sastādītāji ir patiesi pateicīgi Dr. inž. J. Eidukam (RTU), Dr. dat. J. Ķikutam (LU Matemātikas un informātikas institūts), Dr. inž. G. Matisonam (RTU), A. Mauliņai (Latvijas Nacionālā bibliotēka), prof., Dr. h. inž. L. Niceckim (RTU), Dr. fiz. G. Šlihtai (Fizikālās enerģētikas institūts) un Maģ. dat. A. Vasiļjevam (SIA Tilde) par aktīvu līdzdalību LZA Terminoloģijas komisijas informātikas apakškomisijas darbā un viņu ieguldījumu vārdnīcā iekļautās terminoloģijas izstrādāšanā. Īpaši pateicīgi vārdnīcas sastādītāji ir prof. V. Skujiņai (Latviešu valodas institūts) ne tikai par aktīvo līdzdalību terminu veidošanā, bet arī par nozīmīgo ieguldījumu vārdnīcas galīgās redakcijas pilnveidošanā.

Neskatoties uz patiesu vēlēšanos un pūlēm rast visveiksmīgākos risinājumus, vārdnīcas sastādītāji apzinās, ka vienmēr var un vajag meklēt iespējas

tālākai latviešu terminoloģijas pilnveidošanai tik dinamiskai attīstībai pakļautā jomā kā datorzinātne. Tāpēc ar interesi gaidīsim jebkuru konstruktīvu piezīmi un priekšlikumu, kas saistīts ar latviešu terminoloģijas pilnveidošanu datorzinātnē.

Piezīmes un priekšlikumus lūdzam sūtīt Dr. dat. G. Fricnovičam LU Elektronikas un datorzinātņu institūtā, Dzērbenes ielā 14, LV-1006 Rīgā (e-pasts: gfricnov@edzi.lza.lv).

## Izmantotā literatūra

Vārdnīcā iekļauto terminu izvēlei, to latvisko ekvivalentu veidošanai un skaidrojumu formulēšanai izmantoti šādi galvenie informācijas avoti:

***A. Latviešu valodā:***


1. Latviešu valodas vārdnīca. Atb. red. D. Guļevska. — Rīga, "Avots". — 1987.
2. V. Skujiņa. Latviešu terminoloģijas izstrādes principi. — Rīga, "Zinātne". — 1993.
3. G. Šlihta, V. Šlihta. Pamati darbam ar personālo datoru. Eksperimentāla mācību grāmata. — Valsts Administrācijas skola, apgāds "Mācību grāmata. — Rīga. — 1994.
4. Datu pārraides un apstrādes sistēmas. Angļu-krievu-latviešu skaidrojošā vārdnīca. Sastādītāji: V. Baumgarts, A. Baums, A. Gobzemis, G. Fricnovičs, I. Ilziņa. — A/s SWH Informatīvās sistēmas. — 1995.
5. Angļu-latviešu vārdnīca. Sastādītāji: Z. Belzēja, I. Birzvalka, L. Jurka, R. Mozere, J. Raškevičs, A. Treilons. — Apgāds "Jāņa sēta", Rīga. — 1995.
6. K. Sataki. Microsoft Word ikvienam. — Rīga: Computerland Rīga.— 1996.
7. Latviešu-angļu vārdnīca. Prof. A. Veisberga redakcijā.— Rīga: SIA Ekonomisko attiecību institūts. — 1997.
8. Datortehnika. — 1994. — Nr. 1—12.
9. Datortehnika. — 1995. — Nr. 1—12.
10. Datortehnika. — 1996. — Nr. 1—9.
11. Datorpasaule. —1996. — okt., nov., dec.
12. Datorpasaule. — 1997. — Nr. 1—4.

***B. Krievu valodā:***

1. В. К. Зейденберг, А. М. Зимарев, А. М. Степанов, Е. К. Масловский. Англо-русский словарь по вычислительной технике. Издание пятое, исправленное и дополненное. — Москва, "Русский язык". — 1989.
2. Архитектура, протоколы и тестирование открытых информационных сетей. Толковый словарь. Под ред. Э. А. Якубайтиса. — Москва, "Финансы и статистика". — 1989.
3. Большой англо-русский политехнический словарь. В двух томах. — Москва, " Русский язык ". — 1991.
4. В. И. Першиков, В. М. Савинков. Толковый словарь по информатике. — Москва, "Финансы и статистика". — 1991.
5. Е. К. Масловский. Англо-русский словарь пользователя персональной ЭВМ. — Московская Международная школа переводчиков, Москва. — 1992.
6. С. Клименко, В. Уразметов. *Internet.* Среда обитания информационного общества. — Российский Центр Физико-Технической Информатики, Протвино. — 1995.
7. Microsoft Press. Толковый словарь по вычислительной технике (Пер. с англ.) — Москва, Издательский отдел "Русская редакция" ТОО "Channel Trading Ltd.". — 1995.
8. Современный англо-русский словарь по вычислительной технике. — Составитель С. Орлов. — Изд. "Лори". — 1996.

***C. Angļu valodā:***

1. ISO/IEC JTC1 / SC21 No. 5934. Collection of Definitions of OSI Vocabulary (April 1991. Version). — 1991.
2. IEEE Standard. 610. Computer Dictionary. Compilation of IEEE Standard Computer Glossarys. — Institute of Electrical and Electronics Engineers, Inc. — 1991.
3. IEEE Std. 610.6 — 1991. IEEE Standard Glossary of Computer Graphics Terminology. — Institute of Electrical and Electronics Engineers, Inc. — 1992.
4. Internetworking Terms and Acronyms. — Cisco Systems, Inc. — 1992.
5. A.Freedman. The Computer Glossary. The Complete Illustrated Desk Reference. Sixth Edition. — AMACOM American Management Association. — 1993.

6. J. M. Rosenberg. Business Dictionary of Computers. — John Wiley & Sons, Inc. — 1993.
7. G. S. Malkin, T. L. Parker. Internet User's Glossary. — Network Working Group. Request for Comments: 1392. FYI:18. — 1993.
8. IBM Dictionary of Computing. Compiled and edited by G. McDaniel. — McGraw-Hill, Inc. — 1994.
9. Computer Dictionary. Second edition. — Microsoft Pressž. — 1994.
10. P. Dyson. The PC User's Pocket Dictionary. Second Edition. — SYBEX Inc. — 1994.
11. S. M. H. Collin. Dictionary of Computing. Second Edition. — Peter Collin Publishing Ltd. — 1994.
12. Webster's New World Dictionary of Computer Terms. Fifth Edition. Compiled by D. Spencer. — MACMILLAN, USA. — 1994.
13. B. Pfaffenberger with D. Wall. QUE's Computer & Internet Dictionary. 6th Edition. — QUE Corporation. — 1995.
14. Dictionary of Multimedia. Ed. S. M. H. Collin. — Peter Collin Publishing Ltd. — 1995.
15. P. E. Margolis. Personal Computer Dictionary. Second Edition. — Random House. — 1996.
16. Oxford Dictionary of Computing. Ed. S. Pyne and A. Tuck. — Oxford University Press. —1996.

# PRIEKŠVĀRDS

## Par vienu valodu vairāk

Mazām tautām ir jāprot par vienu valodu vairāk, un tā ir — pašas tautas valoda. Mazās tautas valoda pati vien neizdzīvos, tā ir īpaši jābalsta. Jauni vārdi rodas tur, kur rodas jauni procesi, lietas, kur atklāj jaunas parādības. Tikai retos gadījumos tas pirmoreiz notiek mazajā tautā, bet ja tauta grib pastāvēt, tai jāspēj savā valodā runāt par visu, kas notiek pasaulē. Būt mazai tautai ir papildu darbs ikvienam šīs tautas pārstāvim, ne tikai radot nosaukumus pasaulē notiekošajam, ko dara nedaudzi, bet arī aktīvi lietojot šos nosaukumus saziņā, kas būtu jādara katram. Šis darbs ekonomiski neatmaksājas, tāpat kā (cik ciniski!) neatmaksājas mazas tautas un valstis. Atmaksa var būt tikai viena — tautas dzīvotspēja. Kam tauta nav pašvērtība, tam arī valoda nebūs svarīga. Ja negribam kopt savu valodu, lai tad Saeima pieņem jaunu valodas likumu, saskaņā ar kuru, teiksim — no 2010. gada, valsts valoda Latvijā būtu angļu valoda. Cik gan daudz problēmu atkritīs! Lai arī cik tas dīvaini liktos, latviešu valodas vismazāk apdraudētākais periods laikam būs bijuši tieši tie 700 nebrīves gadi, kad iekarotāji nāca un gāja, bet ar iekaroto tautu nesajaucās. Piecdesmit gadi pēc otrā pasaules kara iezīmējās ar oficiāli nepaustu kolonizācijas veicināšanu, kas gandrīz noveda latviešus līdz minoritātes statusam pašu zemē un latviešu valodu izskauda no sabiedriskās dzīves. Neatkarības atgūšana izbeidza šo procesu. Protams, nevar neņemt vērā to, ka ievērojama tautas daļa gājusi bojā karos, bēgļu gaitās, nāves nometnēs un izsūtījumā, bet citi latvieši, vairāk vai mazāk piespiesti, nonāca citās tautās un daudzi tur arī palika. Tomēr varētu domāt, ka līdz ar brīvas, demokrātiskas Latvijas valsts veidošanos lielākās briesmas nu ir pāri. Jā, ārējie draudi valodai un tautai ir krietni mazinājušies. Taču pieaug draudi, ko rada tieši demokrātiskas sabiedrības lielā vērtība — brīvība. Visa pasaule ir atvērusies. Katram, it īpaši — jaunam, izglītotam un spējīgam cilvēkam nākas izvēlēties, kur apmesties uz dzīvi un veidot savu karjeru. Latvija ir ļoti maza un ne vienmēr izrādīsies tā piemērotākā vieta. Kas brīvprātīgi aizbrauks no Latvijas, nez vai īpaši kops latvietību. Un pēdējais — kamēr latvietei Latvijā būs tikai viens bērns, nav vērts runāt par latviešu valodas attīstību, jo nav jāgaida necik ilgi, kad nebūs vairs, kam to valodu runāt. Draudi latviešu valodas un tautas pastāvēšanai nu ir tautā pašā.

## Kāds izplatīts mīts

Ir kāds visai izplatīts mīts: latviešu datortermini jāveido no angļu terminiem, par pamatu ņemot to oriģinālizrunu, jo tā, lūk, būs vieglāk saprasties gan ar ārzemniekiem, gan arī savā starpā.

Piebildīsim vēl, ka tā veidot terminus būtu arī visai viegli. Kāpēc tad nesekot šim, atzīsim, racionālajam un pamatotajam priekšlikumam? Tāpēc, ka attīstot šo pašu domu tālāk, neizbēgami nonāksim pie secinājuma, ka vēl ērtāk sazināties Latvijā būtu, lietojot tikai angļu valodu. Bet vēsture liecina, ka ir saglabājušās tikai dažas tautas, kas nerunā savā valodā. Ja nu mums tāda lietu gaita nebūtu pa prātam, darīsim citādi — kārtīgi iemācīsimies angļu, bet turēsim cieņā un attīstīsim arī savējo, latviešu valodu.

## Kāds cits izplatīts mīts

Terminus veido valodnieki, kas nekā nejēdz no lietām un procesiem, ko termini apzīmē, bet tikai mēģina uzspiest mums, profesionāļiem, kaut kādus mākslīgus latvisku terminu izdomājumus.

Nekas nevar būt tālāks no īstenības kā šāds priekšstats. Patiesībā terminus veido nozares speciālisti, bet valodnieki mēdz tos vērtēt no valodas likumību viedokļa un dažkārt piedāvāt iespējamus variantus. Pašiem profesionāļiem taču ir vissvarīgāk saprasties savā starpā, bet tas ir ļoti grūti, ja runā dažādās valodās kā pie Bābeles torņa. Tā kā terminu veidošana ir valodas jaunrade, par maz ir pārzināt savu nozari vien, vēlama arī valodas likumsakarību zināšana. Un protams — griba darīt šo mazās tautas papildu darbu.

## Kā veidot latviešu terminus?

Pārdomām iesākumā sniegsim dažus citējumus, kas noskatīti [1].

"Ja latviešu valodā saskaitītu visus latviskās cilmes un citvalodu cilmes pamatvārdus, tad aizgūto pamatvārdu būtu apmēram 40—50% no visa pamatvārdu krājuma." [2].

"..Svešvārdu vārdnīcā (R., 1978) ievietoto aizguvumu analīze rāda, ka apmēram 80% no visiem t.s. svešvārdiem ir latīņu un grieķu cilmes internacionālismi. Un tikai 20% aizgūto vārdu ir dota norāde par to aizgūšanu no konkrētas mūsdienu nacionālās valodas, to skaitā 8,74% — no franču, 3,12% — no angļu, 2,42% — no vācu. Tālāk seko aizguvumi no itāļu, spāņu, arābu valodām, bet septīto astoto vietu dala aizguvumi no krievu un holandiešu

valodas (pa 0,44%)." [1]. Jāpiezīmē, ka reālais anglismu procents mūsdienās strauji aug.

"Par produktīvu mūsdienu internacionālismu devēju tiek atzīta angļu valoda. Tomēr attieksme pret anglicismiem vairākās valodās mēdz būt noraidoša. Par objektīvu faktoru, kas negatīvi ietekmē anglismu iekļaušanu internacionālismu kategorijā, jāatzīst angļu valodas fonētiskās sistēmas īpatnība — izrunas un rakstības atšķirība, jo saskaņā ar internacionālismu definīciju to izrunai un rakstībai dažādās valodās jābūt līdzīgai. Salīdzināsim: angļu *byte, design, jersey* un latviešu *baits, dizains, džersija*, kas ar angļu vārdiem identificējami runas līmenī, nevis rakstos. Savukārt vācu *Design* ar angļu *design* identificējami rakstos, nevis runā." [1].

"Katrai valodai, kamēr tā ir veselā stāvoklī, ir dabiska tieksme novērst no sevis svešo, ja tas ir ielauzies — izstumt to no jauna vai vismaz pielīdzināt pašu valodas elementiem. Nav tautas, kas spētu attīstīt un lietot visas skaņas, un katra valoda vairās no tām, kuras tai ir neraksturīgas un pretīgas. Tas, kas attiecināms uz skaņām, vēl jo vairāk attiecas uz vārdiem." [3].

"Neviena valoda nevar iztikt ar terminelementiem, kas veidoti no pašu valodas vārdu saknēm. Valodas attīstības procesā visu laiku notiek vārdu aizgūšana no citām valodām. Tas ir neizbēgams process. Tomēr ieviest citvalodu terminus bez īpašas nepieciešamības nevajag." [4].

Aprobežosimies šoreiz ar šiem, tāpat jau diezgan apjomīgajiem citējumiem. Tomēr lasītājs var iebilst, ka praktiskai vārddarināšanai būtu vēlams kāds īss un koncentrēts vadlīniju kopojums, vēlams — ar piemēriem. Tiešām, arī LZA Terminoloģijas komisijas Informātikas apakškomisija nonāca pie tāda paša atzinuma un nu jau vairākus gadus ievēro kopīgā pieredzē izkristalizējušos 10 principus. Vēloties iesaistīt terminoloģijas darbā pēc iespējas vairāk speciālistu, kā arī izskaidrot savas darbības metodi, minēsim šeit šos "baušļus".

## Teminoloģijas veidošanas vadlīnijas

1. Latviešu valodai raksturīgs fonētiskais alfabēts, t.i. "kā runā, tā raksta" un otrādi. Tāpēc valodā labi iekļaujas aizguvumi no šajā ziņā līdzīgās latīņu valodas, citvalodu aizguvumi ar latīnisku izcelsmi, piemēram, *verifikācija, implementācija, programmatūra,* kā arī, dažkārt, transliterēti aizguvumi no citām valodām, ko konkrētajā gadījumā "kā runā, tā raksta", piemēram, angliskie *serveris, ploteris, ports*.

2. Pieļaujama citvalodu terminu pārcelšana latviešu valodā arī "pēc izrunas", t.i., transkribējot. Tomēr šādā gadījumā īpaša vērība pievēršama valodas labskaņai, kā arī jāmēģina piedāvāt (vismaz kā sinonīmu) latviskas izcelsmes terminu. Tā radušies, piemēram, *saskarne — interfeiss, dzinis — draiveris, datne — fails.*

3. Atšķirīgiem terminiem izcelsmes valodā būtu jāatbilst atšķirīgiem terminiem latviešu valodā. Tāpēc, piemēram, angliskajiem *security, reliability, safety* kā atbilstoši doti *drošība, uzticamība* un *nebīstamība*, nevis mēģināts "lāpīties" ar vienas saknes *droš* atvasinājumiem.

4. Latviskā termina "atpakaļtulkojumam" jādod nepārprotami tas pats izcelsmes valodas termins. To panākt var vai nu "sapārojot" ikdienas leksikas pamatnozīmes vārdu (piemēram, *mouse — pele*), vai arī veidojot latviešu jaunvārdu (piemēram, *menu — izvēlne, prompt — uzvedne*).

5. Jaundarināmo terminu nedrīkst piedāvāt atrauti no apkārtnes, no tam tuviem un analoģiskiem terminiem, kā arī no jau lietotiem terminiem. Jāpārliecinās, vai vārds "labi strādā", t.i., vai tas ir lokāms un ērti lietojams vārdu savienojumos, vai no tā saknes veidojams lietvārds, darbības vārds, īpašības vārds u.tml. Piemēram, latviskais *skaitļotājs* atkāpās latīniskā *dators* priekšā arī tāpēc, ka no tā neērti veidot atvasinājumus (teiksim, darbības vārda *datorizēt* veidošana neizsauc grūtības, bet kā to pašu atvasināt no *skaitļotājs?*).

6. Praksē jau plaši lietotus terminus bez pietiekama pamatojuma mainīt nevajadzētu. Piemēram, lai arī ne vienam vien patīk *skanneris* (no angļu *scanner*), tomēr jau sen tiek lietots *skeneris*. Nav vēlams *skaneris*, jo tā tiktu radīta vēl viena nozīme darbības vārdam *skanēt*.

7. Latviskas izcelsmes vārdiem dodama priekšroka.

8. Terminiem, kas ir vai būs ikdienas leksikas sastāvdaļa, jāpievērš īpaša vērība. Tiem jābūt īsiem (Miķelis Svilans no Otavas pat saka — ja nevar izveidot 2—3 zilbīgu latviskas cilmes terminu, tad tas arī nav jādara), trāpīgiem un labskanīgiem. Šauri profesionāliem terminiem prasības nav tik stingras.

9. Latviešu valoda tiek uzskatīta par skanīgu valodu, t.i., tajā patskaņi un līdzskaņi ir pietiekamā līdzsvarā. Šo līdzsvaru nevajag izjaukt, piesārņojot valodu ar neraksturīgiem skaņu sakopojumiem (piemēram, skaņu sakopojums *pjū* vārdā *kompjūters*).

10. Neviena no iepriekšminētām vadlīnijām nav absolutizējama.

Un tā — šie "baušļi" ir mūsējie. Kādam citam tie var būt pavisam citi. Tādā atšķirībā nav nekā slikta, tikai tā ir skaidri jāapzinās.

## Atsauces

1. V. Skujiņa. Latviešu terminoloģijas izstrādes principi. — Rīga, 1993. 224 lpp.
2. A. Ahero. Par latviešu valodas aizguvumiem // Latviešu valodas kultūras jautājumi. — Rīga, 1967. —3. laid. — lpp. 43.
3. Программа словаря братьев Гриммов, составленная Яковом Гриммом. — Я. Трот. Филологические разыскания. — 1899. — Н. 1. — Стр. 146 —182.
4. Краткое методическое пособие по разработке и упорядочение научно-технической терминологии. — Москва, 1979. — 127 с.

**1 A**
**A**
**A**

Identifikators, ko kā *MS-DOS*, tā arī citas operētājsistēmas piešķir diskdzinim, kurā ir starta instrukcijas.

**2 abort [of a process]**
**priekšlaicīgā [procesa] pārtrauce**
**преждевременное прекращение [процесса]**

Programmas, komandas, procedūras vai datu pārraides pārtraukšana gadījumā, ja netiek ievēroti sistēmai noteiktie ierobežojumi. Priekšlaicīgas pārtrauces var iniciēt operētājsistēma vai process, kura sekmīga pabeigšana nav iespējama. Tās var izsaukt arī kļūda programmā, barošanas sprieguma svārstības vai atslēgšanās, kā arī citi neparedzēti cēloņi.

**3 absolute address**
**absolūtā adrese**
**абсолютный адрес**

Adrese, kas brīvpieejas atmiņā tieši norāda operanda atrašanās vietu, neprasot tās izskaitļošanu.

**4 accelerator key**
**paātrinājumtaustiņš**
**быстрая клавиша**

Vienlaicīgi nospiežamu tastatūras taustiņu kombinācija, kas nodrošina noteiktas funkcijas izpildi, kura, izmantojot peli, jāmeklē izvēlnē.

**5 acceptance criteria**
**akceptēšanas kritēriji**
**критерии приемки**

Kritēriji, kas sistēmai vai tās komponentam jāapmierina, lai to pieņemtu lietotājs, pasūtītājs vai kāda cita pilnvarota juridiska vai fiziska persona.

**6 acceptance test**
**akcepttests**
**тест приемки**

Programmatūras vai aparatūras galīgā pārbaude, kas parāda izstrādātā produkta funkcionālās iespējas. Šī pārbaude nosaka, vai pārbaudāmais objekts darbojas atbilstoši līgumā fiksētajām prasībām (specifikācijai), t.i., vai tas atbilst akceptēšanas kritērijiem.

**7 access method**
**pieejas metode**
**метод доступа**

Datora operētājsistēmas vai datora tīkla vadības programmas sastāvdaļa, kas veic datu saglabāšanu un izguvi vai pārraidīšanu un saņemšanu. Šo programmu funkcijās parasti ietilpst arī aparatūras vai tīkla disfunkciju radītu datu pārraides traucējumu atklāšana un, iespēju robežās, to novēršana.

**8 access path**
**pieejas ceļš**
**путь доступа**

1. Noteiktā secībā izkārtoti datu bāzes ieraksti, kas jāizskata lietojumprogrammai kādas operācijas izpildes gaitā.
2. Datu pārraides tīklos — mezglu secība, caur kuriem tiek pārraidīts nosūtāmais ziņojums.
3. Ceļš, ko secīgi izskata operētājsistēma, lai atrastu kādas datnes izvietojumu datora atmiņā. Pieejas ceļš parasti sākas ar diska apzīmējumu un pa direktoriju (apakšdirektoriju) ķēdi noved pie datnes vārda. Šo terminu izmanto,

strādājot ar firmas *IBM* un ar tiem saderīgajiem, kā arī ar *Macintosh* saimes datoriem.

**9 access protocol**
**pieejas protokols**
**протокол доступа**
Priekšrakstu kopa, kas nosaka stacijas darbību, kad tā veic informācijas pārraidi pa lokālā tīkla koplietošanas datu pārraides vidi.

**10 access server**
**pieejas serveris**
**сервер доступа**
Dators, kas attāliem lietotājiem, kuri parasti pievienoti lokālajam tīklam ar modemu starpniecību, nodrošina iespēju izmantot tīkla resursus tā, it kā lietotāja dators būtu tieši ieslēgts tīklā.

**11 access time**
**pieejas laiks**
**время доступа**
Laika intervāls no informācijas pieprasīšanas brīža datora pamatatmiņai līdz brīdim, kad informāciju saņem tā ierīce, kas to pieprasījusi.

**12 accessory**
**piederums**
**принадлежность**
Datoram pievienota ārējā vai cita ierīce (piem., pele vai modems), kas paplašina tā funkcionālās iespējas, bet nav obligāti nepieciešama datora darbībai.

**13 action message**
**darbības ziņojums**
**сообщение к действию**
Ziņojums, kas informē lietotāju, ka tam nepieciešamās darbības izpilde tiek atlikta, jo noticis kaut kas neparedzēts vai var notikt kaut kas nevēlams. Lietotājs var turpināt vajadzīgo darbību, ignorējot šo ziņojumu, vai veikt nepieciešamās korekcijas, kā arī atlikt tās izpildi vai griezties pēc palīdzības. Terminu parasti izmanto firmas *IBM* vispārējā lietotāja pieejas arhitektūrā. Sk. arī *information action*, *warning action*.

**14 active database**
**aktīvā datu bāze**
**активная база данных**
Datu bāze, ko konkrētā laika brīdī izmanto lietotājs un kas šajā laikā atrodas datora brīvpieejas atmiņā.

**15 active hub**
**aktīvais centrmezgls**
**активный концентратор**
Centrmezgls, kas pastiprina tīklā pārraidāmos signālus, tādējādi ļaujot tos sūtīt starp stacijām, kuras atrodas viena no otras lielākā attālumā, nekā tas būtu pieļaujams pasīvā centrmezgla gadījumā.

**16 active matrix display**
**aktīvās matricas displejs**
**дисплей с активной матрицей**
Augstas kvalitātes šķidro kristālu displeju izveidošanai izmantota tehnoloģija, kas katram displeja ekrāna punktam piesaista aktīvu tranzistoru, lai uzlabotu attēla kontrastainību. Sk. arī *passive matrix display*.

**17 active page**
**aktīvā lappuse**
**активная страница**
1. Lappuse, kas datora operatīvajā atmiņā var tikt modificēta vai parādīta displeja ekrānā.
2. Personālajā datorā ar krāsu videoadapteri — ekrānbufera lappuse, kurā

lietotājs var ierakstīt datus, kamēr tā tiek parādīta displeja ekrānā kā vizuālā lappuse.

**18 active window**
**aktīvais logs**
**активное окно**

Logu vidē par aktīvu sauc logu, ar kuru pašreizējā brīdī strādā lietotājs. Aktīvajā logā ir informācija par izpildāmo uzdevumu, un tas var saņemt datus un lietotāja komandas. Atšķirībā no neaktīvā loga tā virsrakstjosla parasti ir īpaši iezīmēta (tumšāka, gaišāka, citā krāsā). Sk. *windows environment*.

**19 ad hoc query**
**ekspromtvaicājums**
**незапланированный запрос**

Vaicājums, ko izmanto kādam atsevišķam vai specifiskam nolūkam. Parasti šādu vaicājumu izmanto tikai dažas reizes un pēc tam atmet.

**20 adapter**
**adapteris**
**адаптер**

Ierīce, kas nodrošina sadarbību starp atšķirīgām iekārtām un sistēmām, kuru konstruktīvās un funkcionālās īpatnības neļauj izmantot tiešu savienojumu.

**21 adaptive maintenance**
**adaptīvā uzturēšana**
**адаптивное сопровождение**

Programmatūras uzturēšana, kas nodrošina datora programmu izmantošanu mainītā vidē.

**22 add-in program**
**pievienojumprogramma**
**дополнительная программа**

Programma, kas izstrādāta, lai, izmantojot lietojumprogrammas, varētu paplašināt to funkcionālās iespējas.

**23 address**
**adrese**
**адрес**

Unikāls identifikators, ko izmanto, lai viennozīmīgi lokalizētu datora atmiņas šūnas, ziņojuma, datora tīkla mezgla vai cita objekta atrašanās vietu. Sk. arī *addressing*.

**24 address space**
**adrešu telpa**
**адресное пространство**

Kopējais atmiņas apjoms, ko var izmantot programmas. Tā, piem., personālajam datoram, kas izveidots, lietojot firmas *Intel* mikroprocesoru 80386, izmantojamās fizikālās atmiņas apjoms var sasniegt 4 gigabaitus, bet virtuālās — 64 terabaitus.

**25 addressing**
**adresēšana**
**адресация**

Objekta atrašanās vietas identificēšanas metode. Izšķir vienkāršo un hierarhisko adresēšanu. Vienkāršās adresēšanas gadījumā tiek norādīta tikai objekta atrašanās vieta. Biežāk izmanto hierarhisko adresēšanu, kad adreses apvieno grupās. Šāds grupējums norāda objektu savstarpējos sakarus, kā arī atspoguļo datoru tīklu topoloģiju un ir saistīts ar maršrutēšanu. Adresēm var būt fiksēts vai mainīgs garums.

**26 Advanced Interactive eXecutive (AIX)**
**operētājsistēma *AIX*, *AIX***
**операционная система *AIX***

Firmas *IBM* operētājsistēmas *UNIX*

versija, kas paredzēta personālajiem datoriem, darbstacijām, minidatoriem un lieldatoriem, kā arī *RS/6000* sērijas datoriem ar *RISC* procesoru.

**27 Advanced super Thin-layer and high Output Metal Media (ATOMM)**
**tehnoloģija *ATOMM***
**технология *ATOMM***

Japānas firmas *Fuji* diskešu izgatavošanas tehnoloģija, kas ļauj 3,5 collu disketē ierakstīt datus, kuru apjoms var sasniegt 100 megabaitus.

**28 Advanced Technology Attach ment Packet Interface (ATAPI)**
**tehnoloģijas *AT* piesaistes paketes saskarne, saskarne *ATAPI***
**интерфейс *ATAPI***

Standarts, kas atvieglo lasāmatmiņas kompaktdiska diskdziņa pievienošanu resursdatoram, izmantojot saskarni *EIDE*.

**29 Advanced Techology (AT)**
**tehnoloģija *AT***
**технология *AT***

Tehnoloģija, ko firma *IBM* izmantoja, izstrādājot uz mikroprocesora *Intel 80286* bāzes personālo datoru *PC/AT*, kurš atšķirībā no *PC/ XT* klases datoriem ir 3—4 reizes ātrāks. Šim datoram bija jauna tastatūra, 1,2 megabaitu lokanais disks un 16 bitu datu kopne. Sk. arī *AT key board*, *AT bus*, *PC AT*.

**30 agent**
**aģents**
**агент**

Fona programma, kuras darbību parasti izsauc kāds īpašs notikums. Tā, piem., aģents var pārraudzīt ieejas datus un brīdināt lietotāju par to saņemšanu.

**31 AIX**

Sk. *Advanced Interactive eXecutive*.

**32 alert box**
**trauksmes lodziņš**
**окно предупреждений**

Neliels lodziņš, kas parādās displeja ekrānā, lai brīdinātu lietotāju par potenciāliem darbības traucējumiem, piem., par iespēju, ka sistēma var izdzēst vienu vai vairākas datnes. Atšķirībā no dialoglodziņa trauksmes lodziņa parādīšanās neprasa nekādu lietotāja reakciju. Lai atbrīvotu ekrānu no šī lodziņa, lietotājam, nospiežot ievadīšanas taustiņu vai noklikšķinot peles pogu, tikai jāapstiprina, ka trauksmes lodziņš ir pamanīts. Trauksmes lodziņš ir viens no ziņojumlodziņa veidiem.

**33 alias**
**aizstājvārds**
**имя-псевдоним**

Vārds, ar kuru datortīklā tiek apzīmēta persona vai personu grupa. Datoru sistēmās šo terminu izmanto kā datņu vai datora ierīču simbolisku apzīmējumu.

**34 alignment**
**līdzināšana**
**выравнивание строки**

1. Personālajos datoros precīza lasīšanas/rakstīšanas galviņu novietošana virs celiņiem, no kuriem jālasa vai kuros jāieraksta dati.
2. Teksta (teksta bloka) novietošana tā, lai tas piekļautos formas (lappuses) malām vai būtu centrēts tās vidū. Līdzināšana var būt horizontāla vai vertikāla.

Sk. arī *flush left, flush right, justification.*

**35 Alpha AXP**
**arhitektūra *Alpha AXP***
**архитектура *Alpha AXP***
Personālajiem datoriem, darbstacijām un superskaitļotājiem paredzēta firmas *DEC RISC* procesora slēgta arhitektūra, kurā izmantota superskalārā instrukciju izpilde, fiksēta un peldoša komata procesori, kā arī datu un instrukciju kešatmiņas.

**36 alpha test**
**alfa tests**
**альфа-тестирование**
Jaunizstrādātas programmatūras vai aparatūras pirmais pārbaudes posms, ko veic paši izstrādātāji. Sk. arī *beta test.*

**37 alphanumeric data**
**burtciparu dati**
**алфавитно-цифровые данные**
Dati, kuru pierakstam izmantotas kāda alfabēta rakstzīmes: burti, cipari vai citi simboli. Sk. arī *American Standard Code for Information Interchange, character set, Extended Binary Ccoded Decimal Interchage Code.*

**38 alphanumeric display**
**burtciparu displejs**
**алфавитно-цифровой дисплей**
Displejs, kurš nodrošina burtu, ciparu un citu rakstzīmju vizuālu izvadi.

**39 Alt key**
Sk. *Alternate key.*

**40 alternate key**
**alternatīvā atslēga**
**альтернативный ключ**
Jebkura kandidātatslēga datu bāzē, kurai nav piešķirta primārās atslēgas loma.

**41 Alternate key (Alt key)**
**alternēšanas taustiņš**
**альтернативная клавиша**
Firmas *IBM* un ar to saderīgas tastatūras taustiņš, kuru nospiežot vienlaicīgi ar citiem taustiņiem, tiem tiek piešķirta cita, alternatīva nozīme, piem., kādas funkcijas izsaukums.

**42 American Standard Code for Information Interchange (ASCII)**
**Amerikas informācijas apmaiņas standartkods, kods *ASCII, ASCII***
**американский стандартный код для обмена информацией, код *ASCII***
Septiņu bitu kods, ko rakstzīmju kopas elementu identificēšanai izmanto vairumā personālo datoru, daudzos videoterminālos un telekomunikāciju sistēmās. Kods sākotnēji izstrādāts kā ASV nacionālais standarts, bet vēlāk tas kļuvis arī par starptautisku standartu. Koda *ASCII* standartvariantā katrai rakstzīmei, ieskaitot paritātes kontroles bitu, atbilst 8 biti (1 baits). Tomēr ievērību guvis arī paplašinātais kods *ASCII*, kurā izmanto visas 256 koda kombinācijas. Tas ļauj kodēt dažu valodu specifiskos burtus, matemātiskos simbolus un citas rakstzīmes.

**43 analog monitor**
**analogmonitors**
**аналоговый монитор**
Monitors, kas ļauj attēlot neierobežotu noteiktas krāsas spilgtuma diapazonu — no visspilgtākā līdz pilnīgai šīs krāsas

izslēgšanai no attēla. Sk. arī *digital monitor*.

**44 analog output card**
**analogās izvades karte**
**карта аналогового вывода**
Karte, kas nepieciešama, lai pārveidotu datora izvades cipardatus analogajā formā.

**45 anchor**
***enkurs***
**якорь**
Iezīmēts hiperteksta apgabals. Ja uz šo apgabalu norāda kursors, tad peles klikšķis izsauc dokumentu, kura atrašanās vietu nosaka hipersaite.

**46 animation**
**animācija**
**мультипликация**
Kustības ilūzijas radīšana, izveidojot datorā virkni nedaudz atšķirīgu attēlu, kurus ātri rādot displeja ekrānā iegūst nepartrauktas kustības iespaidu.

**47 anonymous FTP**
**datņu anonīmās pārsūtīšanas protokols, anonīmais *FTP***
**анонимный *FTP***
Plaši izmantojama protokola *FTP* versija, kas ļauj iegūt informāciju no tīkla *Internet* arhīviem bez lietotāja identificēšanas un slepenas paroles uzrādīšanas. Vēršoties pie servera ar anonīmā *FTP* starpniecību, lietotājs sava identifikatora vietā uzrāda vārdu *'anonymous'*, bet par paroli izmanto vārdu *'guest'* (viesis). Anonīmais *FTP* neļauj lietotājam ierakstīt jaunas datnes vai modificēt esošās.

**48 ANSI.SYS**
**draiveris *ANSI.SYS***
**драйвер *ANSI.SYS***
Draiveris, ko *MS-DOS* lietotāji var izmantot ekrāna un tastatūras vadībai, piem., lai attīrītu ekrānu no nevajadzīgajiem simboliem vai piekārtotu komandas funkcionālajiem taustiņiem.

**49 anticipatory paging**
**apsteidzošā lapošana**
**страничный обмен с упреждением**
Atmiņas iedalīšanas tehnika, kurā, paredzot lappušu nepieciešamību, tās jau iepriekš tiek pārsūtītas no palīgatmiņas uz operatīvo atmiņu. Sk. arī *demand paging*.

**50 anti-virus program**
**pretvīrusu programma**
**антивирусная программа**
Programma datora vīrusu atklāšanai un likvidēšanai. Ja vīrusu nav iespējams likvidēt, bojāto programmu iznīcina. Sk. arī *computer virus, vaccine*.

**51 API**
Sk. *Application Program Interface*.

**52 APPLE**
Sk. *Apple Computer, Inc.*

**53 Apple Computer, Inc.**
**firma *Apple Computer*, firma *Apple***
**фирма *Apple Computer***
Viena no pazīstamākajām ASV personālo datoru ražotājām firmām. Pirmos personālos datorus *Apple II* uz mikroprocesoru *ROCKWELL 6502* bāzes šī firma sērijveidā sāka ražot 1977. g. As-

toņdesmitajos gados firmas izstrādājumu klāstā parādījās *Macintosh* saimes 32 bitu un *Portable Macintosh* personālie datori. Šie datori nav saderīgi ar *IBM PC*, jo tajos izmantoti nevis firmas *Intel*, bet *Motorola 68000* saimes mikroprocesori, kā arī atšķirīgi datņu formāti un datorkopnes. 1992. g. firmas *Apple*, *IBM* un *Motorola* kopīgi izstrādāja arhitektūru *POWER PC* un firma *Apple* uzsāka ražot personālos datorus *Power Macintosh 6100*, *7100*, *8100*, izmantojot šai arhitektūrai atbilstošus mikroprocesorus (piem., *PC 601*, *PC 604* u.c.).

**54 Apple II**
**personālais dators *Apple II***
**персональный компьютер *Apple II***

Firmas *Apple* personālo datoru saime, kas tiek uzskatīta par mikrodatoru attīstības pirmsākumu un ko plaši lietoja skolās un sadzīvē. Saimes nosaukums radies no 1977. g. izstrādātā pirmā firmas *Apple* sērijveidā ražotā datora apzīmējuma. Vēlākajos gados šo saimi papildināja virkne dažādu pirmā datora uzlabotu modifikāciju (*Apple IIc*, *Apple IIe*, *Apple IIgs*, *Apple III*).

**55 Apple Talk**
**tīkls *Apple Talk***
**сеть *Apple Talk***

Relatīvi vienkāršs un lēts lokālais tīkls, ko izstrādājusi firma *Apple*, lai ļautu apvienot vienā tīklā dažādu firmu personālos datorus. Tīkla *Apple Talk* protokolu sistēma veidota atbilstoši atvērto sistēmu sadarbības bāzes etalonmodelim. Tīklā *Apple Talk* izmantota daudzpieejas metode ar nesēja jušanu un sadursmju nepieļaušanu (*CSMA/CA* metode).

**56 applet**
**sīklietotne**
**специальная прикладная программа**

Neliela lietojumprogramma, kas paredzēta kāda specifiska uzdevuma izpildei (piem., kalkulatora programma, kāršu spēles programma u.c.).

**57 application**
**lietojums; lietotne**
**приложение; прикладная программа**

1. Datora izmantošana īpašiem nolūkiem, piem., algu saraksta sastādīšanai, inventāra uzskaitei, aviobiļešu rezervēšanai u.c. vajadzībām.
2. Programma, ko izmanto, lai ar datora palīdzību izpildītu noteikta veida darbus. Šo terminu bieži izmanto terminu "lietojumprogramma" un "lietojumprogrammatūra" vietā.

**58 application development system**
**lietojumprogrammu izstrādes sistēma, lietotņu izstrādes sistēma**
**система разработки прикладных программ**

Programmu izstrādes līdzekļu komplekts, kas paredzēts lietojumprogrammu izstrādei. Šī programmu pakotne parasti sastāv no redaktora, kompilatora un atkļūdotāja. Tā var ietvert arī bieži lietojamu palīgprogrammu bibliotēku, ko izmanto jaunu programmu izstrādei.

**59 application generator**
**lietojumģenerators**
**генератор прикладных программ**

Programmatūras izstrādāšanas sistēma, kas pēc tam, kad programmētājs ir no-

teicis lietojumprogrammas funkcijas, ģenerē tai atbilstošus pirmkodus (mašīnas komandas). Lietojumģeneratori tiek iekļauti dažās datu bāzēs, un tie izmanto iebūvētu instrukciju kopas, lai ģenerētu programmu kodus. Sk. arī *application development system*.

**60 application icon**
**lietojumikona**
**пиктограмма прикладной программы**
Neliels attēls vai grafisks simbols, kas apzīmē lietojumprogrammu grafiskajā lietotāja saskarnē.

**61 application object**
**lietojumobjekts**
**прикладной объект**
Forma, ar kuru lietojumprogramma nodrošina lietotāju, piem., izklājlapa. Šo terminu parasti izmanto firmas *IBM* izstrādātajā kopējā lietotāju pieejas arhitektūrā. Sk. arī *user object*.

**62 application option**
**lietojumopcija**
**прикладная опция**
Izvēle, kuru programmētājs var realizēt lietojumprogrammā. Šo terminu parasti izmanto firmas *IBM* izstrādātajā kopējā lietotāju pieejas arhitektūrā. Sk. arī *user option*.

**63 application program**
**lietojumprogramma**
**прикладная программа**
Sk. *application*.

**64 Application Program Interface (API)**
**lietojumprogrammu saskarne, saskarne *API***
**интерфейс прикладных программ**
Lietojumprocesos izmantojama pilna operētājsistēmas funkciju specifikācija, kā arī šo funkciju izmantošanas procedūru apraksts. Tīkla operētājsistēmās ar saskarni *API* tiek definēta standarta metode, kas lietojumiem nodrošina visu tīkla iespēju izmantošanu.

**65 application software**
**lietojumprogrammatūra**
**прикладное программное обеспечение**
Sk. *application*.

**66 application window**
**lietojumlogs**
**окно прикладной программы**
Grafiskajā lietotāja saskarnē — galvenais lietojumu logs, kas sastāv no virsrakstjoslas, lietojumu izvēlņu joslas un darba laukuma. Darba laukumā parasti ir viens vai vairāki dokumentu logi.

**67 application-oriented language**
**lietojumorientēta valoda**
**язык, ориентированный на конкретное применение**
Konkrētam lietojumam paredzēta datorvaloda, piem., valoda datora aparatūras projektēšanai.

**68 Archie**
**sistēma *Archie*, *Archie***
**система *Archie***
Tīkla *Internet* datņu meklēšanas sistēma. Sistēmas *Archie* serveri automātiski savāc un piegādā lietotājam ziņas par tīklā atrodamo informāciju, periodiski pieslēdzoties visiem publiski izmantojamajiem protokola *FTP* serveriem. Ar *Archie*

komandu palīdzību lietotājs var ērti uzzināt, kur atrodas viņu interesējošās datnes.

**69 archieval backup**
**arhīvdublējums**
**архивация**

Dublēšanas procedūra, kuras izpildes gaitā visas cietajā diskā esošās datnes ar as programmas starpniecību arhivācijas vajadzībām tiek pārkopētas disketē, magnētiskajā lentē vai citā datu vidē.

**70 architectural design**
**arhitektoniskā projektēšana**
**архитектурное проектирование**

Aparatūras un programmatūras komponentu un to saskarņu noteikšana datoru sistēmas struktūras izstrādāšanai. Sk. arī *functional design*.

**71 archive**
**arhīvs**
**архив**

1. Reti lietojamu programmu un datu, kā arī to uzkrāšanai (saglabāšanai) paredzētās programmatūras apvienojums datu vidē. Arhīvā var saglabāt, piemēram, sistēmas žurnālus, programmu versijas un dublētājkopijas.
2. Datoru tīklā glabātu datņu kopums. Tīklā *Internet* parasti šīs datnes ir pieejamas ar protokola *FTP* starpniecību.

**72 ARCnet**
**tīkls *ARCnet***
**сеть *ARCnet***

Tīklošanas arhitektūra un lokālo tīklu ierīču un programmatūras komplekss, kura sākotnējo versiju 60. gadu beigās izstrādājusi firma *Datapoint Corporation*. Uz šīs arhitektūras bāzes samērā ērti var veidot lētus zvaigžņveida topoloģijas tīklus, kas apvieno plašu personālo datoru un darbstaciju klāstu. Tīklos *ARCnet* izmanto vides pieejas vadības protokolu, kas ir līdzīgs marķiermaģistrāles tīklu marķiera nodošanas procedūrai.

**73 array**
**masīvs**
**массив**

Sakārtots vienāda tipa (loģiski viendabīgs) elementu kopums. Viena masīva elementi var būt veseli skaitļi, cita — reāli skaitļi, bet vēl cita — rakstzīmju virknes. Katram masīva elementam vienozīmīgi piekārtota indeksu kopa (apakšraksts), kas nosaka tā vietu masīvā. Datu apstrādē plaši lieto viendimensijas masīvus vai vektorus un divdimensiju masīvus vai matricas, kuru elementiem attiecīgi piekārtoti viens vai divi indeksi.

**74 array processor**
**matricprocesors**
**матричный процессор**

Savstarpēji savienotu identisku procesoru grupa, kas darbojas sinhroni un ir paredzēta vienlaicīgai vairākdimensiju masīvu elementu apstrādei. Sk. arī *vector processor*, *math coprocessor*.

**75 arrow keys**
**bulttaustiņi**
**клавиши управления курсором**

Tastatūras četru taustiņu grupa, kas pārvieto kursoru vai rādītāju pa labi, pa kreisi, uz augšu un uz leju.

**76 article**
**e-raksts**
**статья**

Elektroniskā pasta ziņojums, ko nosūta

vienai vai vairākām tīkla *USENET* intereškopām un kas ir pieejams katram intereškopas dalībniekam un tīkla *Internet* lietotājam.

### 77 ASCII

Sk. *American Standard Code for Information Interchange*.

### 78 assembler
### asamblers
### ассемблер

Programma, kas pārvērš asamblervalodā rakstītas instrukcijas mašīnvalodas komandās (objektkodos).

### 79 assembly language
### asamblervaloda
### язык ассемблера

Programmēšanas valoda, kuras komandu struktūru nosaka mašīnvalodas komandu un datu formāti, kā arī datora arhitektūra. Asamblervalodā pieļaujamas arī augstāku līmeņu valodu konstrukcijas: izteiksmes, literāļi un makroinstrukcijas.

### 80 assignment statement
### piešķires priekšraksts
### оператор присваивания

Pamata operators visās programmēšanas valodās (izņemot deklaratīvās valodas), kas mainīgajam piešķir jaunu vērtību.

### 81 asterisk
### zvaigznīte
### звездочка

Tastatūras simbols, ko vairākas operētājsistēmas, arī *MS-DOS*, izmanto kā aizstājējzīmi vienas vai vairāku rakstzīmju vietā. Programmēšanas valodās un lietojumprogrammās ar zvaigznīti apzīmē reizināšanas darbību.

### 82 Asynchronous Transfer Mode (ATM)
### asinhronās pārsūtīšanas režīms, *ATM* režīms, *ATM*
### режим асинхронной передачи

Informācijas parsūtīšanas metode, ko izmanto platjoslas *ISDN* tīklos. *ATM* specifikācijā definētas fiksēta garuma (53 baitu) paketes (šūnas), kuras raida, izmantojot fizikālā kanāla asinhrono laikdales multipleksēšanu. *ATM* režīms paredzēts tīkliem, kuros pārraides ātrums sasniedz vairākus gigabitus sekundē.

### 83 AT

Sk. *Advanced Technology*.

### 84 AT bus
### kopne *AT*
### шина *AT*

Kopne, ko firma *IBM* 1984. g. ieviesa *PC/AT* personālajos datoros. Šo kopni sauc arī par izvēršanas kopni. Kopne *AT* atšķirībā no *IBM PC* oriģinālās kopnes, kas nodrošina 8 bitu vienlaicīgu pārraidi, ļauj vienlaicīgi pārraidīt 16 bitu datus. Sk. arī *PC bus*, *Extended Industry Standard Architecture*, *Industry Standard Architecture*, *Micro Channel Architecture*.

### 85 AT keyboard
### tastatūra *AT*
### клавиатура *AT*

Firmas *IBM* personālo datoru 84 taustiņu tastatūra, kas vēlāk paplašināta līdz 101 vai 102 taustiņiem. Atšķirībā no tastatūras *XT* tai ir 12 funkcionālie taustiņi, kas izvietoti vienā rindā tastatūras augšdaļā, kā arī atsevišķa cipartastatūra, kas atrodas tastatūras *AT* labajā pusē.

### 86 ATAPI

Sk. *Advanced Technology Attachment Packet Interface*.

**87 ATM**
Sk. *Asynchronous Transfer Mode.*

**88 ATOMM**
Sk. *Advanced super Thin-layer and high Output Metal Media.*

**89 atribute**
**atribūts**
**атрибут**
Informatīvs elements, kas apzīmē kādu īpašību. Izšķir dažāda veida atribūtus. Datnes atribūts norāda datnes raksturu (lasāmdatne, apslēptā datne, sistēmas datne) vai informē par izmaiņām, kas izdarītas pēc pēdējā datnes dublējuma. Displeja ekrāna atribūts nosaka fona vai priekšplāna krāsas vai citas displeja ekrānā redzamā attēla (teksta) īpašības (piem., fontu maiņu, pasvītrojumus u.c.). Datu bāzēs par atribūtu parasti sauc ieraksta lauka nosaukumu vai tā struktūru.

**90 audiotex**
**audiotekss**
**аудиотекс**
Interaktīva datora un lietotāja sadarbības sistēma, kas paredz, ka lietotājs uz jautājumu, ko tam balsī pa telefonu uzdevis dators, atbild, nospiežot sava telefona tastatūras taustiņu.

**91 audit trail**
**auditācijas pieraksts**
**контрольный след**
Ieraksti atmiņā, kas izdarīti dažādās informācijas apstrādes procesa stadijās, lai šos ierakstus varētu vēlāk noteiktā secībā pārskatīt un izsekot aplūkojamā procesa norisi.

**92 authentication of user**
**lietotāja autentificēšana**
**аутентификация пользователя**
Procedūra, kas vairāklietotāju sistēmās un datoru tīklos ļauj pārbaudīt, vai lietotājs atbilst uzrādītajam identifikatoram. Par identifikatoriem var kalpot, piem., dažādas paroles, šifrēšanas atslēgas u.c. Lietotāja autentificēšanu izmanto, lai aizsargātu datu apstrādes sistēmas no nesankcionētas pieejas, reģistrētu lietotāju un izvēlētos tam atbilstošu apkalpošanas režīmu.

**93 authoring**
**autorēšana**
**авторизация**
Process, kurā kāda konkrēta informācija tiek attēlota multividē, izmantojot ne tikai tekstu, bet arī skaņu, grafiku un videokomponentus.

**94 authoring language**
**autorēšanas valoda**
**язык авторизации**
Augsta līmeņa programmēšanas valoda datorizēta mācību kursa programmatūras izveidošanai, izmantojot multivides iespējas. Sk. arī *authoring system.*

**95 authoring system**
**autorēšanas sistēma**
**система авторизации**
Programmatūra, ko parasti izmanto multivides lietojumos. Autorēšanas programma ar speciālu komandu starpniecību nodrošina lasāmatmiņas kompaktdisku atskaņotāju vadību, kā arī skaņu datņu un videoklipu atskaņošanu. Sk. arī *authoring language.*

**96 authorization**
**pilnvarošana, sankcionēšana**
**санкционирование**
Pilnvaru piešķiršana personai vai personu grupai noteiktu darbību izpildei datu apstrādes sistēmā.

97 **AUTOEXEC.BAT**
**datne *AUTOEXEC.BAT***
**файл *AUTOEXEC.BAT***
Specializēta fona datne, kas automātiski tiek izmantota, tikko dators startē vai restartē.

98 **autorepeat key**
**pašatkārtošanās taustiņš**
**клавиша с автоматическим повтором**
Tastatūras taustiņš, kas nodrošina automātisku kādas rakstzīmes atkārtošanos, kamēr šis taustiņš ir nospiests.

99 **autosave**
**automātiska saglabāšana**
**автоматическое сохранение**
Programmas spēja automātiski saglabāt datni, dublējot to pēc kāda noteikta laika intervāla vai pēc noteikta tastatūras taustiņu nospiešanas reižu skaita.

100 **autotracing**
**autotrasēšana**
**автотрассировка**
Rastrgrafikas attēlu pārveidošana vektorgrafikas attēlos.

101 **auxiliary storage**
**palīgatmiņa**
**вспомогательная память**
Sk. *secondary storage.*

102 **availability**
**izmantojamība**
**готовность**
Sistēmas vai tās komponenta spēja konkrētos apstākļos izpildīt uzdotās funkcijas noteiktajā laika brīdī vai laika posmā, pieņemot, ka pieprasītie ārējie resursi tiek nodrošināti.

101 **auxiliary storage**
**palīgatmiņa**
**вспомогательная память**
Sk. *secondary storage.*

102 **availability**
**izmantojamība**
**готовность**
Sistēmas vai tās komponenta spēja konkrētos apstākļos izpildīt uzdotās funkcijas noteiktajā laika brīdī vai laika posmā, pieņemot, ka pieprasītie ārējie resursi tiek nodrošināti.


103 **background**
**fons**
**фоновая работа**
1. Darbs, programma vai process, ko izpilda reizē ar citiem darbiem, programmām vai procesiem un kam attiecībā pret citiem darbiem, programmām vai procesiem parasti ir zemāka prioritāte.
2. Displeja ekrāna vai tā daļas, uz kuras attēlotas rakstzīmes (piem., melnas rakstzīmes uz balta ekrāna vai otrādi), iekrāsojums.

104 **backlit**
**aizmugurgaismojums**
**задняя подсветка**
Šķidro kristālu displeju apgaismošanas veids, kas piezīmjdatoros un klēpjdatoros uzlabo displeja lasamību un ko īsteno, iebūvējot gaismas avotu displeja ekrāna aizmugurē. Sk. arī *sidelit.*

105 **backspace**
**atpakaļatkāpe**
**возвращение**

Kursora kustība par vienu pozīciju pa kreisi, izdzēšot šajā pozīcijā ierakstīto rakstzīmi.

106 **Backspace key**
**atkāpšanās taustiņš**
**клавиша возврата на позицию**
Tastatūras taustiņš, kas izdzēš rakstzīmi pa kreisi no kursora. Dažās sistēmās atkāpšanās taustiņš izdzēš arī iezīmēto tekstu.

107 **backup**
**dublējums**
**резерв**
Programmas, diska satura vai datnes dublikāts, ko gadījumā, ja aktīvie dati var tikt bojāti vai iznīcināti, izmanto arhīva veidošanai vai vērtīgu datu drošai saglabāšanai.

108 **backup copy**
**dublētājkopija**
**резервная копия**
Lietojumprogrammu vai datņu kopija, ko izveido arhivācijas vajadzībām vai aktīvās kopijas bojājuma gadījumam, lai nodrošinātos pret datu pazaudēšanu.

109 **backward compatible**
**atpakaļsaderīgs**
**обратно-совместимый**
Saderīgs ar kāda produkta agrākajiem modeļiem vai versijām. Par atpakaļsaderīgu sauc produkta jaunu versiju, ja tā var funkcionēt, izmantojot tos pašus datus, ko agrākās šī produkta versijas.

110 **BAK file**
***BAK* datne**
***BAK* файл**
Operētājsistēmās *MS-DOS* vai *OS/2* automātiski vai ar kādas lietojumprogrammas palīdzību izveidotas dublētājdatnes nosaukuma paplašinājums.

111 **bank**
**banka**
**банк**
Identisku aparatūras komponentu kopums. Sk. arī *memory bank*.

112 **bank switching**
**banku komutācija**
**коммутация банков памяти**
Atmiņas paplašināšanas veids, veicot ātru divu atmiņas banku komutāciju. Banku komutācija dod iespēju pārvarēt operētājsistēmas vai mikroprocesora adrešu ierobežojumus.

113 **bar chart**
**joslu diagramma**
**столбчатая диаграмма**
Sk. *bar graph*.

114 **bar code**
**svītrkods**
**штриховой код**
Šaurāku un platāku svītru salikums, kas attēlo noteiktu kodētu informāciju un ko parasti izmanto dažāda veida produkcijas apzīmēšanai. Svītrkods ir ērti apstrādājams datorā un to nolasa, izmantojot lāzera staru vai ierīci, kas satur gaismas avotu un fotošūnu.

115 **bar graph**
**joslu grafiks**
**гистограмма**
No horizontālām vai vertikālām joslām veidots grafiks, ko izmanto, lai salīdzinātu divas vai vairākas vērtības noteiktā laika momentā, piem., bankas procenta likmi atkarībā no depozīta.

116 **barrel**
**muciņa**
**бочкообразное искажение**
Attēla kropļojums, kam raksturīgs attēla paplašinājums vidusdaļā un sašaurinājums tā augšējā un apakšējā daļā.

117 **baseline**
**bāzlīnija**
**базовая линия**
1. Specifikācija vai produkts, ko formāli novērtē un akceptē, lai izmantotu to kā pamatu turpmākajām izstrādnēm. Bāzlīniju var mainīt tikai noteiktas procedūras.
2. Dokuments vai dokumentu kopa, kas izveidota un fiksēta noteiktā kādas konfigurācijas dzīves cikla posmā.
3. Horizontāla izlīdzināšanas līnija mazajiem burtiem (piem., a, u, v), kuriem nav tādu lejupejošu daļu, kas atrodas zem šīs līnijas (piem., p, q, j, y).

118 **BASIC (Beginner's All purpose Symbolic Instruction Code)**
**valoda *BASIC***
**язык *BASIC***
Viegli apgūstama un lietojama programmēšanas valoda, kas domāta darbam dialogrežīmā un ko izmanto visu tipu datoros.

119 **Basic Input/Output System (BIOS)**
**ievadizvades pamatsistēma, sistēma *BIOS*, *BIOS***
**базовая система ввода-вывода**
Programmu kopa, kas parasti ierakstīta firmas *IBM* vai ar tiem saderīgu personālo datoru lasāmatmiņā. Šīs programmas nodrošina datora darbības uzsākšanu un veic diskdziņu, tastatūras, monitoru un citas aparatūras vadību.

120 **BAT file**
***BAT* datne**
***BAT* файл**
Operētājsistēmas *MS-DOS* datnes nosaukuma paplašinājums, ko izmanto fona datņu identifikācijai. Šo datņu komandas tiek secīgi izpildītas.

121 **batch file**
**pakešdatne**
**командный файл**
Specializēta datne, ar secīgi vienu pēc otras izpildāmu operētājsistēmas komandu sarakstu. Operētājsistēmā fona datnei ir paplašinājums *BAT*.

122 **benchmark**
**etalonuzdevums**
**эталонная задача**
Standartizēts tests datu apstrādes sistēmas aparatūras un programmatūras veiktspējas noteikšanai. Pārbaudes procesā pie zināmas slodzes nosaka sistēmas raksturlielumus, kurus vēlāk salīdzina ar analoģisku sistēmu, kas pakļauta līdzīgai pārbaudei.

123 **beta software**
**beta programmatūra**
**бета-программы**
Programmatūras versija, kuras testēšana pirms nodošanas pasūtītājam nav pabeigta un var saturēt kādu kļūdu. Sk. arī *beta test*.

124 **beta test**
**beta tests**
**бета-тестирование**
Jaunizstrādātās programmatūras vai aparatūras otrais pārbaudes posms, kas tiek veikts pasūtītājorganizācijā normālā darba režīmā. Sk. arī *alpha test*.

**125 binding**
**saistīšana**
**привязка**

Process, kura gaitā tiek apvienoti divi informatīvi elementi. Visbiežāk šo terminu lieto, lai piekārtotu, piem., fizisko adresi loģiskajai adresei vai vērtību kādam mainīgajam.

**126 BIOS**

Sk. *Basic Input/Output System*.

**127 bit (binary digit)**
**bits**
**бит**

Informācijas daudzuma mērvienība, kas atbilst vienai binārā skaitļa pozīcijai. Ar bitu skaitu parasti raksturo datorā apstrādājamās informācijas vienības: baitus, vārdus, pusvārdus, dubultvārdus, kilobaitus un megabaitus.

**128 bit map**
**bitkarte**
**побитовое изображение**

Atmiņas apgabals, kurā ierakstīts videoattēls. Vienkrāsas displejiem bits bitkartē attēlo vienu ekrāna pikseli. Pelēkuma skalas un krāsu displejiem pikseli attēlo ar vairākiem bitkartes bitiem.

**129 bit-mapped font**
**bitkartēts fonts**
**шрифт с побитовым отображением**

Noteikta lieluma un stila rakstzīmju kopa, kurā katra rakstzīme aprakstīta kā unikāla bitkarte — punktu šablons. Lai bitkartētu fontu attēlotu uz ekrāna vai izvadītu uz printera un ierakstītu datora vai printera atmiņā, jābūt saglabātam katras rakstzīmes pilnam attēlam. Sk. arī *downloadable font, outline font, scalable font*.

**130 bit-mapped graphics**
**bitkartēta grafika**
**графика с побитовым отображением**

Sk. *raster graphics*.

**131 BITNET**
**tīkls *BITNET***
**сеть *BITNET***

Teritoriāls datoru tīkls, kas apvieno vairākus tūkstošus universitāšu un pētniecības centru lieldatorus Ziemeļamerikā, Eiropā un Japānā. Tīkls *BITNET* neizmanto protokolu *TCP/IP*, bet var apmainīties ar elektroniskā pasta ziņojumiem ar tīklu *Internet*. Attīstoties tīklam *Internet*, tīkls *BITNET* pakāpeniski zaudē savu nozīmi.

**132 bits per second (bps)**
**biti sekundē**
**биты в секунду**

Binārā kodētās informācijas pārraides ātruma mērvienība — bitu skaits sekundē.

**133 black hole**
**melnais caurums**
**черная дыра**

Hipersaite globālajā tīmeklī *WWW*, kas norāda uz dokumentu, kurš ir izdzēsts vai pārvietots citā vietā.

**134 blank**
**tukšums**
**пробел**

1. Datu vides daļa, kurā nav ierakstītas rakstzīmes.
2. Datorgrafikā — visa attēla vai tā daļas neparādīšana displeja ekrānā.

135 **blank cell**
**tukšā šūna**
**пустая ячейка**

Izklājlapas šūna, kurā nav ne skaitlisku lieluma, ne iezīmes, ne makrodefinīcijas, ne formulas.

136 **blank character**
**tukšumzīme**
**знак пробела**

1. Rakstzīme, ko izmanto programmu un datu pierakstā, lai norādītu atstarpi gan starp programmas elementiem, gan starp datiem.
2. Grafisks atstarpes rakstzīmes attēlojums.
3. Rakstzīme, kas veido tukšumu grafisku rakstzīmju virknē.

137 **blinking**
**mirgošana**
**мерцание**

Displeja ekrānā redzamas rakstzīmes vai attēla uzzibsnīšana un apdzišana. Mirgošana ir tipiska kursoram, iespraušanas punktiem, izvēlnēm un brīdinājumu ziņojumiem, jo tā piesaista lietotāja uzmanību.

138 **block**
**bloks**
**блок**

1. Iekārtas elementu kopums, kas veic noteiktas funkcijas un kas konstruktīvi izveidots kā atsevišķs funkcionāls mezgls.
2. Bitu vai rakstzīmju grupa, ko apstrādā vai pārsūta kā vienotu veselumu.
3. Datora atmiņā — ierakstu kopums, ko apstrādā vai pārsūta kā vienotu veselumu. Sk. arī *data block*.

139 **block move**
**bloka pārvietošana**
**перемещение блока**

Operācija, ko veic tekstapstrādes sistēmās, lai pārvietotu teksta bloku no vienas dokumenta vietas citā. Sk. arī *cut and paste*.

140 **BMP file**
***BMP* datne**
***BMP* файл**

Datnes vārda paplašinājums, kas norāda, ka datnē ir operētājsistēmas *Microsoft Windows* standartam atbilstoši bitkartētas grafikas attēli.

141 **board**
**plate**
**плата**

Sk. *printed circuit board*.

142 **boilerplate**
**tekstveidne**
**библиотека стандартных текстов**

Teksta fragmenti (standartfrāzes, paragrāfi, lappuses), ko atkārtoti izmanto, veidojot dažādus dokumentus (vēstules, pārskatus, līgumus u.c.). Tekstveidni parasti ievieto dokumentā bez izmaiņām vai ar nelielām modifikācijām (piem., pievienojot adresāta personiskos datus — vārdu, uzvārdu, adresi u.c.).

143 **bold**
**treknraksts**
**жирный шрифт**

Burtstils, kurā rakstzīmes veidotas tumšākas, izmantojot biezinātus elementus.

144 **bold italic**
**treknais slīpraksts**
**жирный курсив**

Burtstils, kurā rakstzīmes biezinātas un ieslīpinātas pa labi.

**145 boldface**
**treknā druka**
**жирная печать**

Burtstils, kas teksta daļu, kurā tas izmantots, padara tumšāku un biezāku par pārējo tekstu. Trekno druku parasti izmanto kādas teksta daļas izcelšanai.

**146 bookmark**
**grāmatzīme**
**закладка**

1. Kods, ko ievieto kāda dokumenta noteiktā punktā, lai atvieglotu tā atrašanu. Grāmatzīmi, piem., var ievietot kāda teksta punktā, kurā nepieciešams atgriezties, vai kādā bieži izmantojamā datnes vietā.
2. *WWW* pārlūkprogrammā paredzēta iespēja saglabāt norādi uz visbiežāk izmantojamajiem *WWW* resursiem, kas nodrošina tiem ātru un ērtu pieeju. Sk. arī *Uniform Resource Locator*.

**147 boot**
**sāknēšana**
**самозагрузка**

Datora darbības iniciēšana, kuras gaitā pēc operatīvās atmiņas attīrīšanas tajā tiek ierakstīta operētājsistēma. Personālajam datoram pēc tā ieslēgšanas šī darbība parasti notiek automātiski.

**148 bootstrap**
**sāknēšanas programma**
**программа самозагрузки**

Īsa programma, kas pastāvīgi glabājas datora energoneatkarīgajā atmiņā un kas vajadzības gadījumā no lasāmatmiņas operatīvajā atmiņā izsauc uzdevumu risināšanas vadības sistēmu (piem., operētājsistēmu) vai tās ielādes programmu.

**149 bounced message**
**atlecis ziņojums**
**отскакивающее сообщение**

Elektroniskā pasta ziņojums, kas pēc neveiksmīga mēģinājuma to nodot adresātam atgriežas atpakaļ pie nosūtītāja. Neveiksmīgas nodošanas iemesls var būt kļūdaina adrese vai kļūme datoru tīkla darbībā.

**150 box**
**lodziņš**
**окно**

Grafiskās lietotāja saskarnes terminoloģijā lodziņš ir norobežots laukums displeja ekrānā, kas atgādina logu. Atšķirībā no pēdējā lodziņu parasti nevar pārvietot, palielināt vai samazināt. Lodziņa parādīšanās ekrānā ir saistīta ar kādas informācijas pieprasījumu vai tās sniegšanu lietotājam. Lodziņu izmanto arī logu vadībai, piem., to lieluma maiņai. Sk. arī *alert box*, *check box*, *combination box*, *dialog box*, *list box*, *message box*, *scroll box*, *zoom box*.

**151 Boyce-Codd normal form**
**Boisa-Kodda normālforma**
**нормальная форма Бойса-Кодда**

Informācijas strukturēšanas veids relāciju datu bāzēs. Normālforma izslēdz redundanci un pretrunīgumu, kā arī nodrošina efektīvu informācijas uzturēšanu, saglabāšanu un atjaunināšanu.

**152 branch**
**zars**
**ветвь**

1. Ceļš kokveida struktūrā, piem., viens vai vairāki viena direktorija apakšdirektoriji operētājsistēmā *MS-DOS*.
2. Algoritmos un programmās — viens

no alternatīvajiem to izpildes variantiem, kuru izvēlas atkarībā no zināmu noteikumu kopuma.
3. Tīklu topoloģijā — ceļš, kas savieno divus blakus mezglus un nesatur nevienu starpmezglu.

153 **Break key**
**pārtraukšanas taustiņš**
**клавиша прерывания**
Firmas *IBM* un ar to saderīgas tastatūras taustiņš, kas parasti izvietots tastatūras augšpusē un ir apvienots ar pauzēšanas un ritslēga taustiņu. Ja vienlaicīgi ar šo taustiņu tiek nospiests vadīšanas taustiņš, tad tiek ievadīta pārtraukumkomanda. Tā parasti liek datoram pārtraukt komandas vai programmas izpildi.

154 **brouter**
**tiltmaršrutētājs**
**мост-маршрутизатор**
Ierīce, kurā kombinētas tilta un maršrutētāja funkcijas. Tā izpilda vienu no šīm funkcijām atkarībā no tā, kādas protokolu sistēmas ir realizētas tīklos, ko apvieno ar šīs ierīces starpniecību.

155 **browse**
**pārlūkot**
**просмотреть**
Izskatīt datu bāzi vai datņu sarakstu, lai atrastu kādu konkrētu objektu. Pārlūkošanu parasti veic, lai ar informāciju iepazītos, nevis lai to mainītu.

156 **browser**
**pārlūkprogramma, pārlūks**
**программа ускоренного просмотра**
Lietojumprogramma, kas paredzēta datu bāzu, datņu sarakstu, kā arī *WWW* dokumentu izskatīšanai, lai atrastu lietotājam vajadzīgo informāciju. Pārlūkprogrammas var būt grafiskas vai tekstuālas. Sk. arī *browsing*.

157 **browsing**
**pārlūkošana**
**просмотр**
1. Datu bāzes vai datņu saraksta izskatīšana, lai atrastu kādu konkrētu objektu.
2. Hiperteksta dokumentu izskatīšanas process, kurā izmantotas hipersaites.

158 **brush**
**ota**
**кисть**
Krāsošanas programmas rīks attēlu veidošanai un zīmējumu laukumu aizpildīšanai ar krāsu. Krāsošanas programmas var izveidot dažāda platuma otas vilcienus.

159 **buffer**
**buferis**
**буфер**
Atmiņa īslaicīgai datu glabāšanai, ko pārsūta starp divām ierīcēm, lai izlīdzinātu to darbības ātrumus, signālu līmeņus vai nodrošinātu asinhronu pārraidi.

160 **bug**
**blusa**
**ошибка**
Kļūda, kas programmētāja vai shēmtehniķa neuzmanības dēļ rodas programmas vai shēmas sastādīšanas procesā. Šis termins neattiecas uz kļūdu, kas pieļauta, formulējot projekta uzdevumu.

161 **bullet**
**aizzīme**
**маркер позиции**
Simbols (parasti aplis vai kvadrāts) teksta rindiņas sākumā, kas norāda, ka ar šo

rindiņu sākas kāda īpaša teksta daļa vai kāda saraksta elements.

**162 Bulletin Board System (BBS)**
**ziņojuma dēļa sistēma,**
**sistēma *BBS*, *BBS***
**система публикации**
**объявлений**

Sistēma datorā, kuru izmanto lietotāju grupa, kam ir kopīgas intereses, lai apmainītos ar informāciju. Parasti šo sistēmu izmanto, lai lietotāji ar telefona līniju un modemu starpniecību varētu nodibināt savstarpējos sakarus. Lietotājs var pārskatīt saņemtos ziņojumus, atstāt ziņojumus citiem interesentiem, kā arī sazināties ar citiem lietotājiem.

**163 burst mode**
**sprādzienrežīms**
**пакетно-монопольный режим**

Datu pārraides metode, kas nodrošina datu savākšanu un ātru pārraidi vienotā blokā. Sk. arī *Micro Channel Architecture*, *PCI local bus*, *VL local bus*.

**164 bus**
**kopne; maģistrāle**
**шина; магистраль**

1. Elektrisko signālu vadītāju (vadu) komplekts, pa kuru notiek informācijas apmaiņa starp dažadiem datora funkcionālajiem blokiem (procesoriem, atmiņu, pieslēgvietām, ārējo iekārtu vadības ierīcēm u.c.). Personālajos datoros parasti izmanto standartizētās kopnes, kó veido trīs dažādu kopņu apvienojums. Pa tām tiek pārsūtīti dati (datu kopne), informācija par datu atrašanās vietu (adrešu kopne) un vadības informācija (vadības kopne). Kopnes galvenokārt raksturo ar bitu skaitu, ko vienlaicīgi var pārsūtīt pa kopni (izšķir 8, 16, 32 utt. bitu kopnes), un ar maksimālo datu ātrumu, ko norāda MHz. Sk. arī *Extended Industry Standard Architecture*, *Industry Standard Architecture*, *Micro Channel Architecture*.
2. Vītais pāris, koaksiālais vai optiskais kabelis, kas lokālajos tīklos savieno datorus un citas iekārtas. Sk. arī *bus network*.

**165 bus arbitration**
**kopņu arbitrāža**
**арбитраж шин**

Pa kopni veicamās datu pārraides vadība un protokols, kas nodrošina saskaņotu kopnes izmantošanas iespēju vairākiem lietotājiem.

**166 bus bridge**
**kopņu tilts**
**шинный мост**

Ierīce, kas datorā savieno divas līdzīgas vai atšķirīgas kopnes.

**167 bus extender**
**kopnes paplašinātājs**
**расширитель шины**

Ierīce, kas pagarina datorkopni vai palielina pa to pārraidāmo bitu skaitu.

**168 bus master**
**kopnes vedējs**
**хозяин шины**

Ierīce, kas vada kopni datu pārraides procesā. Parasti šīs funkcijas veic centrālais procesors. Augstas veiktspējas personālajos datoros kopnes vadībai bieži izmanto tīkla kartē vai videoadapterī iebūvētus palīgprocesorus, kas atbrīvo no šī uzdevuma veikšanas centrālo procesoru.

**169 bus mastering**
**kopnes vešana**
**ведение шины**

Datu pārraides procesa organizēšana kopnē, ko īsteno, pievienojot papildplates, un kas ļauj vadīt datu pārraidi pa kopni neatkarīgi no centrālā procesora, kā arī nodrošināt tiešu pieeju datora atmiņai un ārējām iekārtām. Sk. arī *bus master, bus slave.*

170 **bus mouse**
**kopnes pele**
**шинная мышь**
Pele, kuru, izmantojot speciālu plati, pievieno datora izvēršanas kopnei. Sk. arī *serial mouse.*

171 **bus network**
**maģistrāles tīkls**
**магистральная сеть**
Datoru tīklu topoloģijas variants, kurā visi tīkla mezgli (stacijas) ir pievienoti vienam kabelim, nodrošinot pārsūtītā ziņojuma vienlaicīgu nonākšanu visos tīkla mezglos. Ziņojumu pieņem tikai tas mezgls vai tie mezgli, kuriem ziņojums adresēts. Lai atklātu vai novērstu iespējamās sadursmes, šīs topoloģijas tīklos tiek izmantotas speciālas metodes (piem., *CSMD/CD* vai marķiera nodošanas metode). Šādas topoloģijas variants ir izmantots tīklos *Apple Talk* un *Ethernet*. Sk. arī *ring network, star network.*

172 **bus slave**
**kopnes sekotājs**
**подчиненная шина**
Datu satekne, kas saņem datus no kopnes vedēja.

173 **button**
**poga**
**кнопка**
Fizikāli poga parasti ir kādas norādītājierīces (piem., peles) konstruktīvs elements vai arī tā tiek imitēta displeja ekrānā (ikona). Šādu pogu "nospiež", pārvietojot uz to kursoru un klikšķinot peli. Sk. arī *click, double click, mouse button.*

174 **buzz word**
**liekvārds**
**жужжащее слово**
Bieži sastopams vārds, ko nav lietderīgi īpaši meklēt (piem., vārdi "un", "adrese", "ieraksts" u.tml.). Vairumam datu bāzu ir īpaši liekvārdu saraksti. Veicot nepieciešamās informācijas meklēšanu, šos vārdus atmet.

175 **byte**
**baits**
**байт**
Kā vienots veselums apstrādājama bitu grupa. Baits var būt rakstzīme, komanda vai mašīnvārda daļa. Baits parasti ir datora vismazākā adresējamā atmiņas daļa un sastāv no 8 bitiem.

## C

176 **C (language)**
**valoda C**
**язык C**
Programmēšanas valoda, kas izstrādāta 70. gadu sākumā, lai atvieglotu programmatūras pārnešanu no viena datora uz otru. Tajā apvienotas augsta līmeņa mašīnneatkarīgas valodas un asamblervalodas iespējas. Valoda *C* ir operētājsistēmas *UNIX* pamatvaloda.

177 **C++**
**valoda C++**
**язык программирования C++**

Programmēšanas valodas *C* objektorientēta versija.

**178 CAD**
Sk. *Computer Aided Design.*

**179 Cancel button**
**atcelšanas poga**
**кнопка отмены**
Grafiskās lietotāja saskarnes dialoga lodziņš, ko lietotājs izmanto, lai atsauktu kādas operācijas izpildi un atgrieztos pie apstrādājamā dokumenta. Atcelšanas poga pilda tādas pašas funkcijas kā atsoļa taustiņš.

**180 candidate key**
**kandidātatslēga**
**потенциальный ключ**
Unikāls identifikators, ko relāciju datu bāzē veido viens vai vairāki atribūti. Katrai relācijai ir piekārtota vismaz viena kandidātatslēga (primārā atslēga). Gadījumā, ja šādas atslēgas ir vairākas, viena pilda primārās, bet pārējās — alternatīvo atslēgu funkcijas.

**181 Capitals Lock key**
**burtslēga taustiņš**
**клавиша фиксации**
**верхнего регистра**
Tastatūras taustiņš, kas darbojas kā pārslēgs. Kad burtslēgs ir ieslēgts, tad tastatūras burtu taustiņi ievada lielos burtus, kad izslēgts — mazos burtus.

**182 Caps Lock key**
Sk. *Capitals Lock key.*

**183 caption**
**uzraksts**
**субтитр**
Skaidrojošs teksts loga augšējā pusē, uz ekrāna pogas vai zem attēla.

**184 capture**
**tveršana**
**захват**
Datu iegūšanas (savākšanas) process datorā. Šaurākā nozīmē — displeja ekrānā izspīdināta attēla ierakstīšana datora atmiņā izveidotā datnē vēlākai apstrādei.

**185 card**
**karte**
**карта**
Sk. *printed circuit board.*

**186 carriage return**
**rakstatgrieze**
**возврат каретки**
Vadības rakstzīme, kas kursoram vai printerim dod rīkojumu atgriezties aplūkojamās rindiņas sākumā. Ja saņemta arī rindpadeves rakstzīme, kursors vai printeris vienlaicīgi veic arī nākošās rindiņas padevi.

**187 Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)**
**nesēja jušanas un sadursmju nepieļaušanas daudzpieeja, *CSMA/CA***
**множественный доступ с контролем несущей и предотвращением конфликтов**
Pieejas metode lokālā tīkla staciju kopīgi izmantojamai pārraides videi. Pirms sākt datu pārraidi, stacija nosūta speciālu satrēgumsignālu, lai iegūtu pārraides vidi monopollietošanai un nepieļautu sadursmes derīgās informācijas pārraides gaitā.

**188 Carrier Sense Multiple Access with Collision Detection (CSMA/CD) nesēja jušanas un sadursmju atklāšanas daudzpieeja, *CSMA/CD* множественный доступ с контролем несущей и обнаружением конфликтов**

Pieejas metode vairāku lokālā tīkla staciju kopīgi izmantojamai datu pārraides videi. *Pirms sākt datu pārraidi, stacijas* klausās, vai šajā laikā nenotiek cita pārraide (signāla nesēja jušana). Pārraides laikā *tiek pārbaudīts, vai vienlaicīgi* neraida arī kāda cita stacija. Sadursmes atklāšanas gadījumā notiek atkārtota pārraide.

**189 cartridge kasetne кассета**

1. Slēgts, pārvietojams modulis, kurā ievietota diskatmiņa, magnētiskā lente vai lasāmatmiņas mikroshēma. Sk. arī *disk cartridge, font cartridge, ROM cartridge.*

2. Plastikāta konteiners, kurā iepilda printerī vai kopēšanas iekārtās izmantojamo tinti vai pulveri.

**190 cartridge font kasetnes fonts кассетный шрифт**

Printera fonts, kas ierakstīts lasāmatmiņas kasetnē, kuru var ievietot tai paredzētajā lāzerprintera vai matricprintera ligzdā.

**191 cascaded star kaskādzvaigzne каскадированная звезда**

Tīkla topoloģijas variants ar vairākiem centrmezgliem, kuru savienojumi veido tīkla abonentu hierarhijas līmeņus.

**192 cascading menu kaskādizvēlne меню-каскад**

Izvēlnes sistēma, kurā komandas izvēle izvēršamajā izvēlnē izraisa nākošās izvēlnes parādīšanos. Uz kaskādizvēlnes klātbūtni parasti norāda trīsstūris izvēlnes labajā malā.

**193 cascading windows kaskadēti logi окна каскадом**

Lietotāja saskarnē — divi vai vairāki logi, kas displeja ekrānā ir daļēji pārklājušies. Šāds logu parādīšanas veids ir ērts, jo lietotājs var redzēt visu to logu virsrakstjoslas un malas, ko viņš ir atvēris.

**194 CASE**

Sk. *Computer-Aided Software Engineering.*

**195 cassette kasete кассета**

Noslēgts atmiņas modulis, kas kā datu vidi izmanto magnētisko lenti. Kaseti parasti lieto lētos sadzīves datoros un tajā datus var ierakstīt, lasīt un dzēst.

**196 cathod-ray tube display katodstaru lampas displejs дисплей на электронно-лучевой трубке**

Ierīce, kas datus attēlo vizuālā veidā, izmantojot vadāmus elektronu starus.

**197 CD**

Sk. *Compact Disk.*

**198 CD-I**
Sk. *Compact Disk-Interactive*.

**199 CD-R**
Sk. *Recordable Compact Disk*.

**200 CD-ROM (Compact Disk ROM) lasāmatmiņas kompaktdisks, *CD-ROM* постоянное запоминающее устройство на компактдиске *CD-ROM***
Kompaktdisks, ko izmanto teksta un grafikas, kā arī skaņas ierakstīšanai. Lasāmatmiņas kompaktdisks piemērots lielu datu masīvu uzglabāšanai. Sk. arī *CD-ROM player*, *CD-ROM XA*.

**201 CD-ROM eXtended Architecture (CD-ROM XA) arhitektūra *CD-ROM XA* архитектура CD-ROM XA**
Firmu *Philips*, *Sony* un *Microsoft* 1988. g. izstrādātais lasāmatmiņas kompaktdiska arhitektūras uzlabojums, kas nosaka, kā lasāmatmiņas kompaktdiskā jāuzglabā informācija, lai nodrošinātu vienlaicīgu pieeju kā audioinformācijai, tā arī videoinformācijai. Informācija no šī tipa lasāmatmiņas kompaktdiska var tikt nolasīta, izmantojot standarta *CD-ROM* diskdzini, kas apgādāts ar speciālu arhitektūras *CD-ROM XA* kontrollera karti.

**202 CD-ROM Extensions *CD-ROM* paplašinājumi расширения *CD-ROM***
Programmatūra, kas nepieciešama, lai operētājsistēma (parasti operētājsistēma *DOS*) nodrošinātu *CD-ROM* atskaņotāja izmantošanu. Šī programmatūra lietotājam parasti tiek piegādāta kopā ar atskaņotāju un satur speciālu atskaņotāja draiveri.

**203 CD-ROM player *CD-ROM* atskaņotājs устройство воспроизведения *CD-ROM***
Iekārta informācijas nolasīšanai no kompaktdiskiem un lasāmatmiņas kompaktdiska. *CD-ROM* atskaņotājs tiek pievienots personālajam datoram, izmantojot speciālu kontrollera karti, ko ievieto izvēršanas slotā, tādējādi nodrošinot nolasītās informācijas pārsūtīšanu datoram. Informācijas nolasīšanai un ierakstīšanai kompaktdiska *CD-ROM* atskaņotājā tiek izmantots lāzera stars. Sk. arī *CD-ROM drive*, *CD-ROM Extensions*, *Recordable Compact Disk*.

**204 CD-ROM XA**
Sk. *CD-ROM eXtended Architecture*.

**205 CD-ROM [disk] drive lasāmatmiņas kompaktdiska diskdzinis, *CD-ROM* diskdzinis *CD-ROM* дисковод**
Ierīce, kas ietilpst *CD-ROM* atskaņotājā un parasti veic kompaktdiskā ierakstītās informācijas nolasīšanu un pārsūtīšanu datoram. Atskaņotājs nodrošina diskdziņa griešanos un lāzera stara izveidi kompaktdiskā ierakstīto datu nolasīšanai. Speciāli diskdziņi (piem., *CD-R* diskdzinis) nodrošina arī informācijas ierakstīšanu.

**206 CDDI**
Sk. *Copper Distributed Data Interface*.

**207 cell šūna ячейка**

1. Viena binārā skaitļa glabāšanai paredzēts atmiņas vai reģistra elements ar noteiktu adresi.
2. Rindas un kolonnas krustpunkts izklājlapā.
3. *ATM* režīmā izmantojamas fiksēta garuma paketes.

**208 Central Processing Unit (CPU)**
**centrālais procesors**
**центральный процессор (ЦП)**

Procesors, kas datu apstrādes gaitā veic instrukciju interpretāciju un vada citu datora bloku darbību. Centrālais procesors sastāv no vadības bloka, aritmētiski loģiskā bloka un atmiņas. Personālajos datoros centrālo procesoru parasti veido, izmantojot vienu mikroprocesoru.

**209 Centronics interface**
**firmas *Centronics* saskarne**
**интерфейс фирмы *Centronics***

Firmas *Centronics* izstrādāts *IBM PC* un ar tiem saderīgu personālo datoru paralēlo pieslēgvietu saskarnes standarts. Firmas *Centronics* saskarni parasti izmanto printeru pieslēgšanai datoram.

**210 certification**
**sertificēšana; sertifikācija**
**сертификация**

Formālas pieņemšanas un pārbaudes process vai rakstiskas garantijas izsniegšana, kas apliecina, ka sistēma vai tās komponents atbilst noteiktām prasībām un ir izmantojams paredzēto uzdevumu risināšanai.

**211 CGA**

Sk. *Color Graphics Adapter*.

**212 character**
**rakstzīme**
**знак**

Burts, cipars vai jebkurš cits simbols, kuru izmanto datu grafiskai attēlošanai vai kodēšanai datu apstrādes sistēmās. Rakstzīme datora atmiņā aizņem vienu baitu.

**213 Character-based User Interface (CUI)**
**rakstzīmju orientēta lietotāju saskarne, saskarne *CUI***
**символьно-ориентированный пользовательский интерфейс**

Lietotāja saskarne, kas atšķirībā no grafiskās lietotāja saskarnes komandu ievadīšanai datorā paredz izmantot tastatūru.

**214 character mode**
**rakstzīmju režīms**
**символьный режим**

Sk. *text mode*.

**215 character set**
**rakstzīmju kopa**
**набор знаков**

Datorā izmantojams noteikts rakstzīmju saraksts. Katru rakstzīmi attēlo kā skaitli, piem., koda *ASCII* rakstzīmju kopas attēlošanai tiek izmantoti skaitļi no 0 līdz 127, kuros ietverts angļu alfabēts, cipari un citas speciālas rakstzīmes.

**216 character string**
**rakstzīmju virkne**
**строка знаков**

Lineāri sakārtota rakstzīmju kopa, ko veido jebkura kāda alfabēta galīga garuma rakstzīmju secība, piem., burti, skaitļi, vadības un citas speciālas rakstzīmes.

**217 check box**
**izvēles rūtiņa**
**окно пометки**

Izvēles rūtiņa parasti ir kvadrātveida laukums, kam pierakstīts teksts, kas norāda iespējamo izvēli. Kad lietotājs izdarījis šo izvēli, tad izvēles rūtiņā parādās rakstzīme "x". Var būt viena vai vairākas šādas izvēles rūtiņas. Pēdējā gadījumā lietotājam ir iespējas vienlaicīgi izdarīt vairākas izvēles, piem., izvēlēties burtstilu un burtveidolu.

**218 chip**
**mikroshēma**
**микросхема**

Neliels plāns pusvadītāja (visbiežāk silīcija) kristāls, kurā izveidota integrētā shēma. Termini "mikroshēma" un "integrētā shēma" parasti tiek lietoti kā sinonīmi, lai gan pirmais vairāk akcentē materiālu, bet otrs — izgatavošanas tehnoloģiju un izpildāmās funkcijas.

**219 ciphertext**
**šifrēts teksts**
**шифротекст**

Teksts ar apslēptu semantisko jēgu. Tas iegūts, izmantojot šifrēšanu, un to var atšifrēt tikai tā adresāts, kura rīcībā ir šifra atslēga. Sk. arī *cryptography*, *decryption*, *plaintext*.

**220 circuit**
**shēma**
**схема**

Savstarpēji savienotu elektronisku komponentu kopums, kas veic kādu noteiktu funkciju.

**221 circuit board**
**shēmas plate**
**монтажная плата**

Sk. *printed circuit board*.

**222 CISC**

Sk. *Complex Instruction Set Computer*.

**223 click**
**klikšķis**
**щелчок**

Ātra peles pogas nospiešana un atlaišana pēc tam, kad peles rādītājs novietots uz izvēlētā objekta vai izvēlētajā vietā. Ar klikšķa palīdzību parasti izvēlas vajadzīgo objektu vai novieto kursoru vietā, ko norāda peles rādītājs. Sk. arī *double-click*, *drag select*.

**224 clickable image map**
**klikšķināma attēlkarte**
**отмеченная карта изображений**

Grafisks attēls, kura noteiktas daļas norāda uz hipersaitēm. Novietojot uz šīs attēla daļas kursoru un noklikšķinot peli, pārlūkprogramma atver resursu, uz kuru norāda hipersaite.

**225 client**
**klients**
**клиент**

Iekārta vai lietojumprocess, kam parasti ir tikai viens lietotājs un kas izmanto servera pakalpojumus. Datoru tīklos klients var būt personālais dators vai darbstacija, kas sadarbojas ar datņu serveri saskaņā ar klientservera arhitektūras prasībām.

**226 client/server architecture**
**klientservera arhitektūra**
**архитектура клиент-сервер**

Arhitektūra, kas paredz datu apstrādes procesu sadalīt starp klientu (lietojumprocesu vai personālo datoru, kam parasti ir tikai viens lietotājs) un serveri, kas vienlaicīgi var apkalpot vairākus klientus. Salīdzinot ar terminālsistēmām, šī arhitektūra ļauj daudz efektīvāk organizēt datu apstrādes procesu, iesaistot tajā arī klienta resursus.

227 **clip art**
**klipkopa**
**иллюстративная вставка**
Iepriekš sagatavotu grafisku attēlu kopa, kas glabājas datnē, un ko izmanto teksta apstrādē un datorizdevniecībā. No šīs kopas izvēlas atsevišķus objektus, un tos iesprauž veidojamajos dokumentos.

228 **clipboard**
**starpliktuve**
**буфер вырезанного изображения**
Logošanas vidē rezervēts atmiņas apgabals, kurā kāda dokumenta kopija vai izgriezums no tās (teksts vai grafika) tiek saglabāts, lai to vēlāk ievietotu tajā pašā vai citā dokumentā.

229 **clipping**
**apgriešana**
**отсечение**
Aiz noteiktas robežas (piem., loga rāmja) esošas displeja attēla daļas atdalīšana un atmešana.

230 **clock**
**taktētājs**
**тактовый генератор**
Elektronisks bloks, kas ģenerē secīgus taktimpulsus datora ierīču darbības sinhronizēšanai. Sk. arī *clock/calendar*, *timer*.

231 **clock cycle**
**takts**
**такт**
Laika periods starp diviem viens otram sekojošiem taktimpulsiem.

232 **clock speed**
**taktātrums**
**тактовая частота**
Ātrums (frekvence), ar kādu taktētājs ģenerē taktimpulsus.

233 **clock/calendar**
**pulksteņkalendārs**
**часы-календарь**
Personālajā datorā izveidots iekšējais pulkstenis, kas rāda kā laiku, tā arī mēnesi un gadu. Pulksteņkalendāra nepārtrauktu darbību nodrošina iebūvēta baterija vai akumulators.

234 **clone**
**klonējums**
**клон**
Funkcionāli identiska kādas ierīces, piem., datora, kopija. Klonējums ir aparatūr- un programmsaderīgs ar oriģinālu, bet var no tā atšķirties ārējā izskatā.

235 **closed architecture**
**slēgtā arhitektūra**
**закрытая архитектура**
Arhitektūra, kuras specifikācija nav brīvi pieejama un kuras plašāku izmantošanu īpašniekfirma nepieļauj. Sk. arī *open architecture*.

236 **cluster controller**
**klastera kontrolleris**
**групповой контроллер**
Terminālu grupas vadības ierīce. Klastera kontrolleris saņem komandas no resursdatora un nodod tās attālinātiem terminālļiem, kā arī vada datu apmaiņu starp terminālļiem un resursdatoru.

237 **code**
**kods**
**код**
1. Informācijas viennozīmīgs attēlojums ar kāda alfabēta burtiem, cipariem un citām rakstzīmēm. Kodus izmanto datu

pārveidošanai pirms vai pēc to uzkrāšanas, pārraides vai apstrādes dažādās sakaru un datu apstrādes sistēmās.
2. Programma konkrēta datora valodā.

**238 code page**
**kodu lappuse**
**кодовая страница**

Operētājsistēmā *MS-DOS* (sākot ar versiju 3.3) — tabula, kas nodrošina iespēju lietot dažādās valodās izmantojamu rakstzīmju kopas un tastatūras.

**239 cold boot**
**aukstā sāknēšana**
**холодная самозагрузка**

Datora iedarbināšana, sākot ar barošanas sistēmas ieslēgšanu un operētājsistēmas ielādi.

**240 cold start**
**aukstais starts**
**холодный запуск**

Sk. *cold boot.*

**241 color display**
**krāsu displejs**
**цветной дисплей**

Sk. *color monitor.*

**242 Color Graphics Adapter (CGA)**
**videostandarts *CGA*; video-**
**adapteris *CGA*, adapteris *CGA***
**цветной графический адаптер**
***CGA***

1. Pirmais grafikas videostandarts, ko firma *IBM* izstrādājusi saviem personālajiem datoriem. Teksta režīmā šī standarta augstākā izšķirtspēja 16 krāsām ir 640×200, bet grafikas režīmā 16 krāsām — 160×200, 4 krāsām — 320×200 un 2 krāsām 640×200.
2. Videoadapteris, kas atbilst videostandartam *CGA* un, izmantojot piemērotu monitoru, nodrošina šī standarta prasību izpildi.

**243 color look-up table**
**krāstabula**
**таблица перекодирования**
**цветов**

Vērtību tabula, kurā katrai krāsai, kas glabājas datora atmiņā un var tikt attēlota displeja ekrānā, piekārtots noteikts skaitlisks lielums. Krāstabulas izmantošana paātrina nepieciešamo krāsu izvēli un atvieglo krāsu attēlu veidošanu.

**244 color map**
**krāsu karte**
**карта цветов**

Sk. *color look-up table.*

**245 color monitor**
**krāsu monitors**
**цветной монитор**

Monitors, kas attēla veidošanai displeja ekrānā var izmantot vairāk nekā divas krāsas un to dažādus toņus. Sk. arī *monochrome monitor.*

**246 color printer**
**krāsu printeris**
**цветной принтер**

Vispārējs nosaukums printeriem, kuri veido krāsainus attēlus. Pie šīs printeru grupas pieder, piem., krāsu, strūklprinteri, vaska termopārneses printeri un elektrostatiskie printeri.

**247 column**
**kolonna; aile; sleja**
**колонка**

1. Vertikāla viena pikseļa vai rakstzīmes platuma josla no displeja augšas līdz apakšai.

2. Vertikāla šūnu josla izklājlapās.
3. Vertikāla teksta josla dokumentā.

**248 COM file**
**COM datne**
**COM файл**

*COM* datne ir operētājsistēmas *DOS* vai *OS/2* datne, kuras vārda paplašinājumā ietilpst sufikss "*COM*". Šo sufiksu izmanto, lai apzīmētu datni, kas satur nepārvietojamu, no viena segmenta sastāvošu izpildāmu programmu, kura ietilpst atmiņas apgabalā, kas mazāks par 64 kilobaitiem.

**249 COM port**
**COM pieslēgvieta, COM ports**
**COM порт**

Seriālā pieslēgvieta datu apmaiņai personālajos datoros, kas funkcionē operētājsistēmu *DOS* un *OS/2* vadībā. *COM* pieslēgvietām parasti ir pievienoti modemi vai peles, dažkārt arī printeri. Operētājsistēmas *DOS* versijas līdz 3.2 nodrošina darbu ar *COM1* un *COM2* pieslēgvietām, bet versija 3.3 — arī ar *COM3* un *COM4*. Operētājsistēma *OS/2* dod iespēju izmantot līdz astoņām *COM* pieslēgvietām.

**250 combination box**
**kombinētais lodziņš**
**комбинированное окно**

Lodziņš, kas apvieno ievadlauka un sarakstlodziņa iespējas. Sarakstlodziņš satur izvēles, ko lietotājs var ritināt un atlasīt, lai aizpildītu ievadlauku. Sk. *drop-down combination box*.

**251 command**
**komanda**
**команда**

1. Vadības signāls, kas iniciē noteiktas operācijas izpildi.
2. Instrukcija, kas pārveidota datoram tieši izpildāmā formā.

**252 command button**
**komandpoga**
**командная кнопка**

Spiežampoga grafiskās lietotāju saskarnes dialoga lodziņā, kas iniciē tādas darbības kā izvēlētās komandas izpildi, komandas atcelšanu vai cita dialoga lodziņa izspīdināšanu displeja ekrānā. Sk. arī *Cancel button*, *OK button*.

**253 COMMAND.COM**
**datne COMMAND.COM**
**файл COMMAND.COM**

Datne, kurā ir instrukcijas komandu izpildei. Sk. *command interpreter*.

**254 command-driven program**
**komandvadāma programma**
**программа командного типа**

Programma, kas savas darbības gaitā pieprasa, lai lietotājs ievadītu sintaktiski pareizi formulētas komandas.

**255 command interpreter**
**komandu interpretators**
**интерпретатор команд**

Operētājsistēmā *MS-DOS* — sistēmas programma, kas saņem lietotāja ievadītās komandas, pārveido tās datoram tieši izpildāmā formā un veic nepieciešamās darbības, piem., lietojumprogrammu ielādi, uzvednu un ziņojumu izspīdināšanu displeja ekrānā.

**256 Command key**
**komandtaustiņš**
**командная клавиша**

Taustiņš, kura funkcijas jaunākajās firmas *Apple* tastatūrās ir līdzīgas vadīšanas taustiņa funkcijām firmas *IBM* un ar tām saderīgajās tastatūrās. Nospiežot šo taustiņu kopā ar kādu rakstzīmes taustiņu, var atvieglot izvēles atrašanu izvēļņu hierarhijā (t.i., veikt saīsinājumtaustiņa funkcijas).

**257 command language**
**komandvaloda**
**командный язык**

Augsta līmeņa specializēta valoda ar ierobežotu komandu skaitu, ko tālāk apstrādā komandu interpretators. Kā komandu valodu piemērus var minēt tekstu apstrādes makrovalodas, darbu vadības valodas un datu bāzu vaicājumvalodu.

**258 command line**
**komandrinda**
**командная строка**

1. Vieta ekrānā, kur noteiktā secībā tiek ievadītas komandas.
2. Komandvalodā uzrakstīta teksta virkne, kas tiek nodota izpildīšanai komandu interpretatoram.

**259 command procesor**
**komandu procesors**
**командный процессор**

Operētājsistēmas programma, kas apstrādā lietotāja ievadītās komandas, attēlo displeja ekrānā uzvednes un apstiprinājuma vai brīdinājuma ziņojumus. Šī termina sinonīms ir komandu interpretators.

**260 command prompt**
**komandu uzvedne**
**приглашение на ввод**

Ziņojums, kurš norāda, ka sistēma ir gatava saņemt nākamo komandu. Komandu uzvedne tiek izspīdināta displeja ekrānā kā atsevišķa rakstzīme vai rakstzīmju grupa, piem., kā C:\>.

**261 command separator**
**komandu atdalītājs**
**разделитель команд**

Svītra, kas nodala saistītas komandas izvēlnes logā.

**262 command-line operating system**
**komandrindas operētājsistēma**
**операционная система, работающая в режиме командной строки**

Komandvadāma operētājsistēma, piem., *MS-DOS*, kas prasa komandu ievadi ar tastatūru. Sk. arī *Graphical User Interface*.

**263 comment**
**komentārs**
**комментарий**

Aprakstošs pirmvalodas programmas priekšraksts, ko izmanto programmas dokumentēšanai. Vairumā programmēšanas valodu sintakse pieļauj veidot komentāru, ko programmas kompilēšanas procesā neņem vērā.

**264 Common Communication Support (CCS)**
**kopējais sakaru balstījums**
**единая поддержка коммуникаций**

Protokolu un normatīvu komplekss, kas ir viena no trim firmas *IBM* izstrādātās sistēmu lietojumu arhitektūras sastāvdaļām, kura nodrošina sistēmu un pro-

grammatūru sasaisti. Sk. arī *Common Programming Interface, Common User Access (CUA) architecture.*

**265 Common Programming Interface (CPI)**
**kopējā programmēšanas saskarne, saskarne *CPI***
**единый интерфейс системы программирования**

Lietojumu izstrādāšanas valodu un pakalpojumu kopums, kas ir viena no trim firmas *IBM* izstrādātās sistēmu lietojumu arhitektūras sastāvdaļām, kura nodrošina programmatūras implementāciju ar augstu savstarpējās aizstājamības pakāpi. Sk. arī *Common Communication Support, Common User Access (CUA) architecture.*

**266 Common User Access (CUA) architecture**
**kopējā lietotāju pieejas arhitektūra, arhitektūra *CUA***
**архитектура единого доступа пользователя**

Lietotāja un datora dialoga organizēšanas vadlīnijas, kas ir viena no trim firmas *IBM* izstrādātām sistēmu lietojumu arhitektūras sastāvdaļām, kura nodrošina vienotu lietotāju pieeju datu apstrādes sistēmai. Sk. arī *Common Programming Interface, Common Communication Support*, kā arī *Graphical User Interface.*

**267 communications**
**sakari; komunikācija**
**коммуникация**

1. Mērķtiecīga datu un vadības signālu apmaiņa starp datu avotu un datu saņēmēju, izmantojot sakaru kanālus.
2. Divpusējs informācijas apmaiņas process, kura gaitā saņemtā informācija ir saprotama abiem tā dalībniekiem.

**268 communications channel**
**sakaru kanāls**
**канал связи**

Fizikālie līdzekļi, kas kalpo datu pārraidīšanai starp divām ierīcēm. Datoros izšķir iekšējos un ārējos sakaru kanālus. Iekšējos sakaru kanālus jeb kopnes izmanto datu pārsūtīšanai starp atsevišķām sistēmas daļām, bet ārējos — datora savienošanai ar citiem attāliem datoriem.

**269 communications program**
**komunikācijas programma**
**коммуникационная программа**

Lietojumprogramma, kas ar sakaru līdzekļu palīdzību pārvērš lietotāja izmantojamo datoru terminālī datu pārsūtīšanai un saņemšanai no attāla datora.

**270 communications protocol**
**komunikācijas protokols**
**коммуникационный протокол**

Standartizēta datu apmaiņas procedūra starp datoriem vai datoru un termināli. Tiek izmantoti vienkāršāki un sarežģītāki komunikāciju protokoli. Datņu pārsūtīšanai no viena personālā datora otram var izmantot vienkāršus datņu pārsūtīšanas protokolus (piem., *X*-modems). Sarežģītu komunikācijas protokolu sistēmu veido Starptautiskās standartizēšanas organizācijas septiņu slāņu atvērto sistēmu sadarbības etalonmodelis, kā arī tīklā *Internet* izmantojamie protokoli *TCP/IP*.

**271 compact disk (CD)**
**kompaktdisks**
**компактдиск**

Optiskais disks, kas sākotnēji bija paredzēts augsta blīvuma kvalitatīvai skaņas ierakstei un reproducēšanai ciparsignālu formā. Vēlāk kompaktdiski tiek plaši

lietoti kā lielas ietilpības atmiņas ierīces dažāda rakstura diskrētās informācijas uzglabāšanai. Sk. arī *CD-ROM, CD-ROM player, CD-ROM eXtended Architecture, Compact Disk-Interactive, Digital Video-Interactive, Recordable Compact Disk.*

**272 Compact Disk — Interactive (CD—I)**
**interaktīvais kompaktdisks, *CD—I***
**интерактивный компактдиск**

1. Lasāmatmiņas kompaktdisks, kas paredzēts īpaši formatizētu datu, audio- un videoinformācijas uzglabāšanai un reproducēšanai personālajos datoros un televizoros. Lietotājs interaktīvajā kompaktdiskā ierakstīto informāciju var nolasīt, izmantojot īpašu *CD-I* atskaņotāju.
2. Optisko disku aparatūras un programmatūras standarts, kas apvieno audioinformācijas, videoinformācijas un tekstuālo ierakstu tehnoloģiju lielas ietilpības kompaktdiskiem. Šis standarts aptver informācijas kodēšanas, kompresijas un dekompresijas metodes, kā arī ierakstītās informācijas attēlošanu displeja ekrānā.

**273 Compaq (Compaq Computer Corp.)**
**firma *Compaq***
**фирма *Compaq***

Viena no ievērojamākām personālo datoru ražotājām firmām. 1983. g. firma *Compaq* uzsāka ar *IBM PC* saderīgu datoru ražošanu, bet jau ar 1986. g. savos personālajos datoros sāka izmantot mikroprocesoru *Intel 80386*. Šīs firmas ražojumu klāstā ir piezīmjdatori *Centura*, galddatori *Deskpro*, serveri *ProSigma* un *Presario* ar plašām multivides lietojumu iespējām.

**274 compatibility**
**saderība**
**совместимость**

Īpašību komplekss, kas ļauj darboties vienotā sistēmā dažādu datu apstrādes sistēmu objektiem. Izšķir aparatūrsaderību un programmsaderību. Ar aparatūrsaderību saprot datora spēju sadarboties ar citam datoram paredzētām ierīcēm. Ar programmsaderību saprot iespēju dažādos datoros izmantot vienu un to pašu programmatūru.

**275 compiler**
**kompilators**
**компилятор**

Programma, kas veic kompilēšanu, t.i., augsta līmeņa valodā uzrakstītas programmas translēšanu mašīnvalodā (objektkodā).

**276 Complex Instruction Set Computer (CISC)**
**sarežģītas instrukcijkopas dators, *CISC* dators**
**компьютер со сложным набором инструкций**

Dators (procesors), kurā instrukciju, to izpildes ciklu un adresēšanas veidu skaits atšķirībā no *RISC* datora ir ievērojami lielāks. Šī tipa datoros instrukciju formāts var būt patvaļīgs un to izpildei parasti nepieciešamas vairākas taktis. Sk. arī *Reduced Instruction Set Computer.*

**277 compose sequence**
**kompozītsecība**
**составная последовательность**

Taustiņsitienu secība, ar kuras starpniecību datorā tiek ievadītas rakstzīmes, kas nav atrodamas tā tastatūrā.

278 **composite key**
**kompozītatslēga**
**составной ключ**
Atslēga, kuru veido divi vai vairāki lauki datnē, kolonnas tabulā vai atribūti relāciju datu bāzē.

279 **compound device**
**salikta ierīce**
**составное устройство**
Multivides ierīce skaņas vai cita veida speciālā datnē (piem., *MIDI* datnē) ierakstītās informācijas reproducēšanai.

280 **compound document**
**salikts dokuments**
**соединенный документ**
Teksta datne, kurā ir gan teksts, gan grafika. Atsevišķos gadījumos saliktā dokumentā var būt arī audio- un videoinformācija.

281 **computer**
**dators, skaitļotājs**
**ЭВМ**
Tehniska sistēma (ierīču komplekts), kas saskaņā ar uzdotu programmu veic automātisku datu apstrādi un ievadizvadi. Sk. arī *Complex Instruction Set Computer*, *computer on a chip*, *computer system*, *desktop computer*, *home computer*, *laptop computer*, *microcomputer*, *notebook computer*, *palmtop computer*, *pen computer*, *personal computer*, *pocket computer*, *portable computer*, *Reduced Instruction Set Computer*, *single board computer*.

282 **Computer-Aided Design (CAD)**
**datorizētā projektēšana**
**автоматизированное проектирование**
Datoru izmantošana tehnisku izstrādņu, t.sk. drukāto shēmu un integrēto shēmu projektēšanai. Datorizētas projektēšanas sistēmas parasti strādā dialogrežīmā un izmanto konkrētās nozares objektu izstrādei izveidotu speciālu programmatūru, grafikas ievadizvadi, skenerus un citas perifērijas ierīces.

283 **Computer-Aided Software Engineering (CASE)**
**datorizēta programminženierija**
**автоматизированная разработка программного обеспечения**
Datoru un specializētas programmatūras izmantošana programmu sastādīšanai. Datorizēta programminženierija parasti nodrošina ne tikai šo programmu izstrādāšanu, bet arī sastādāmo programmu funkcionālo pārbaudi, to kodēšanu, testēšanu, nepieciešamās dokumentācijas izveidi un citas ar programmatūras izstrādāšanu saistītās darbības.

284 **computer game**
**datorspēle**
**компьютерная игра**
Izplatīts izklaidēšanās veids, izmantojot datoru interaktīvā režīmā. Datorspēles aptver plašu interešu loku — sākot no alfabēta mācīšanas maziem bērniem līdz šaham, kara spēlēm, kā arī dažādu sadzīves un ekonomikas problēmu risināšanas algoritmu meklēšanai.

285 **computer graphics**
**datorgrafika**
**компьютерная графика**
Informācijas apstrādes režīms, kas ar atbilstošu ievades un apstrādes programmu un ierīču (piem., grafisko displeju, ploteru u.c.) palīdzību ļauj informāciju

attēlot grafiskā veidā. Izšķir divus datorgrafikas pamatveidus: vektorgrafiku jeb objektorientētu grafiku un rastrgrafiku jeb bitkartētu grafiku.

**286 Computer Graphics Metafile (CGM)**
**datorgrafikas metadatne, standarts *CGM, CGM***
**интерфейс компьютерной графики**

Starptautisks grafikas datņu formāta standarts, kas nodrošina objektorientētas grafikas uzglabāšanu aparatūrneatkarīgā formā vai grafikas attēlu apmaiņu starp lietotājiem.

**287 computer on a chip**
**vienkristāla dators**
**однокристальный компьютер**

Mikroshēma, kurā realizēti visi galvenie datora funkcionālie bloki (procesors, brīvpieejas atmiņa, lasāmatmiņa, ievadizvades ierīču kontrolleri u.c.).

**288 computer output microfilmer**
**datora izvades mikrofilmētājs**
**микрофильмовое устройство вывода компьютера**

Izvades ierīce, kas ieraksta datorā veiktās datu apstrādes rezultātus mikrofilmā. Šādu mikrofilmu sauc par datora izvades mikrofilmu.

**289 computer performance evaluation**
**datora veiktspējas novērtēšana**
**оценивание производительности компьютера**

Disciplīna, kas nodarbojas ar datora veiktspējas noteikšanu un pētī metodes, ar kuru palīdzību datora veiktspēju var uzlabot.

**290 computer system**
**datorsistēma**
**вычислительная система**

Datora un tā perifērijas ierīču (t.sk. disku, monitora, dažādu ievadizvades ierīču u.c.) pilna konfigurācija, kas operētājsistēmas vadībā kopīgi veic datu apstrādi.

**291 concentrator**
**koncentrators**
**концентратор**

Ierīce, kas apvieno sakaru kanālus no dažādiem tīkla mezgliem. Koncentrators apvieno grafiku no lēnākiem datu pārraides kanāliem, lai to pārraidītu pa ātrāku kanālu. Atšķirībā no multipleksora, kas tikai sadala ātrākās līnijas kapacitāti starp lēndarbīgākiem abonentiem, koncentrators uzkrāj pienākušos datus pirms to tālākās nosūtīšanas, vajadzības gadījumā veicot tilta, maršrutētāja vai citas funkcijas.

**292 conceptual model**
**konceptuāls modelis**
**концептуальная модель**

Modelis, kas apraksta datu bāzu saturu un struktūru. Konceptuālais modelis atspoguļo datu bāzes ierakstu fizisko un loģisko struktūru, kā arī dod iespēju lietotājam pārskatīt speciāliem lietojumiem nepieciešamās datu bāzes daļas.

**293 concurrency**
**laiksakritība**
**совпадение по времени**

1. Divu vai vairāku procesu (programmu) paralēla izpilde. Šo terminu parasti lieto vienlaicīgi notiekošu procesu aprakstīšanai vairākprocesoru sistēmās.

2. Vienlaicīga datu bāzu resursu izmantošana, kad divi lietotāji mēģina vienā un tajā pašā laikā rediģēt kādu datu bāzes ierakstu vai veikt citu līdzīga rakstura darbību.

**294 concurrency control**
**laiksakritības vadība**
**управление совпадением по времени**

Vairāku vienlaicīgi izpildāmu programmu vai datu apstrādes procesu vadīšana dalītās datu apstrādes sistēmās.

**295 CONFIG.SYS**
**datne *CONFIG.SYS***
**файл *CONFIG.SYS***

Operētājsistēmu *MS-DOS* un *OS/2* konfigurāciju datne. Datne atrodas saknes direktorijā un tajā ir specifikācijas (piem., tastatūras ierīču draiveriem un buferu skaitam). To izmanto, lai ielādētu draiverus un mainītu sistēmas konfigurāciju.

**296 configuration**
**konfigurācija; konfigurēšana**
**конфигурация**

1. Datoru sistēmas komponentu kopums un savstarpējie sakari, ko nosaka šo komponentu tehniskie raksturojumi un risināmā uzdevuma īpatnības.
2. Datoru sistēmas un tās komponentu, kā arī lietojumprogrammu izvietošana un to savstarpējās sadarbības organizēšana, kas pēc iespējas pilnīgāk apmierinātu lietotāju vajadzības. Sk. arī *configuration file*.

**297 configuration file**
**konfigurēšanas datne**
**файл конфигурации**

Datne, ko veido lietojumprogrammas un operētājsistēma un kas ietver informāciju, kuras izmantošana nodrošina datu apstrādes vides (datoru sistēmas, datoru tīkla un to programmatūras) uzstādīšanu un piemērošanu lietotāja vajadzībām.

**298 connector**
**savienotājs**
**соединитель**

Speciāls konstruktīvs elements, kas ar ligzdu vai spraudņu palīdzību ļauj savienot kabeļus, pievienot kabeli ierīcei vai savienot ierīces savā starpā. Sk. arī *female connector*, *male connector*.

**299 consistency**
**nepretrunīgums**
**непротиворечивость**

Dokumentu, sistēmu vai to komponentu vienveidības, standartizēšanas vai saskaņošanas pakāpe. Sk. arī *traceability*.

**300 console**
**pults**
**пульт оператора**

Ierīces, kas nodrošina lietotāja (operatora) sadarbību ar datora operētājsistēmu. Personālajos datoros par pulti parasti sauc tastatūru un displeju.

**301 container**
**konteiners**
**контейнер**

Jebkurš grafiskās lietotāja saskarnes objekts, kas sevī ietver kādu citu objektu. Parastākās konteineru formas ir mape un direktorijs.

**302 context switching**
**konteksta pārslēgšana**
**переключение контекста**

Pārslēgšanās no vienas programmas uz otru, nenobeidzot pirmās programmas izpildi. Konteksta pārslēgšana ļauj lieto-

tājam vienlaicīgi operēt ar vairākām programmām, bet tā atšķiras no vairāk-uzdevumu režīma ar to, ka tiek izmantota tikai viena programma, bet visu citu sistēmā ielādēto programmu izpilde tiek apturēta. Kontekstu pārslēgšanās, izmantojot starpliktuvi, ļauj ātri un ērti informāciju pārnest no vienas programmas uz otru.

### 303 contiguous allocation<br>blakusiedalīšana<br>соседнее распределение

Datora atmiņas iedalīšanas paņēmiens, kas paredz atmiņā glabājamos datus ievietot atmiņas blokos tā, lai loģiski saistītās programmas vai dati arī fiziski atrastos blakus. Sk. arī *paging*.

### 304 control character<br>vadības rakstzīme<br>управляющий символ

Viena no pirmajām 32 koda *ASCII* rakstzīmēm. Katrai no tām ir piekārtota noteikta vadības standartfunkcija, piem., rakstatgrieze, rindpadeve vai atsolis. Lai izpildītu pirmajām 26 rakstzīmēm (A-Z) atbilstošās standartfunkcijas, jānospiež vadīšanas taustiņš un attiecīgās rakstzīmes taustiņš.

### 305 control code<br>vadības kods<br>управляющий код

Viena vai vairākas rakstzīmes, kas noteiktā kontekstā iniciē, modificē vai aptur vadības operāciju izpildi. Vadības kodu parasti nedrukā, un uz displeja ekrāna tas neparādās. Gadījumā, ja tiek izmantota viena rakstzīme (parasti kāda no pirmajām 32 koda *ASCII* rakstzīmēm), šo rakstzīmi sauc par vadības rakstzīmi.

### 306 Control key (Ctrl)<br>vadīšanas taustiņš<br>клавиша управления

Tastatūras taustiņš, kuru nospiežot vienlaicīgi ar citiem taustiņiem, tiem tiek piešķirta cita — alternatīva — nozīme, piem., kādas darbības veikšana. Dažās programmās vadīšanas taustiņš kopā ar atsevišķiem rakstzīmju taustiņiem ievada vadības rakstzīmes. Piemēram, teksta-pstrādes programmās vadīšanas taustiņa un taustiņa L vienlaicīga nospiešana ievada pazīmi pārejai uz jaunu lappusi.

### 307 control menu<br>vadības izvēlne<br>управляющее меню

Izvelkamā izvēlne, ko var atrast praktiski visos logos un dialoglodziņos un kas satur opcijas aktīvo logu pārvaldīšanai. Šīs izvēlnes saturs var mainīties, bet parasti tajā ir komandas logu pārvietošanai, to lieluma maiņai, kā arī logu aizvēršanai, pārslēgšanai uz citas lietojumprogrammas logu vai nākošā dokumenta logu.

### 308 controller<br>kontrolleris<br>контроллер

Ierīce, kas vada datu apmaiņu starp datora centrālo procesoru un citiem tā funkcionālajiem blokiem (atmiņu, displeju, tastatūru, printeri u.c. ārējām iekārtām), atbrīvojot no šī uzdevuma izpildes centrālo procesoru. Personālajos datoros kontrolleri parasti ir to standartkomponentos (displejos, tastatūrā, diskdziņos u.c.) vai izvēršanas platēs iebūvētas mikroshēmas. Tiem jānodrošina datu apmaiņa starp datora izvēršanas kopnēm (piem., kopni *AT*, arhitektūras *MCA* un *EISA* kopnēm).

**309 conventional memory**
**konvencionālā atmiņa**
**базовая память**

Firmas *IBM* un ar tiem saderīgajos personālajos datoros tā parasti ir apakšējā atmiņa, t.i., brīvpieejas atmiņas daļa līdz 640 kilobaitiem, kurai pieeju nodrošina *MS-DOS*.

**310 cooperative processing**
**kooperatīvā apstrāde**
**кооперативная обработка**

Dalītās datu apstrādes sistēmas darbības režīms, kas ļauj diviem vai vairākiem datoriem (piem., lieldatoram un personālajam datoram) vienlaicīgi izpildīt to pašu programmu vai darbu, izmantojot vienus un tos pašus datus. Kooperatīvā apstrāde ir līdzeklis, ar kura starpniecību datoriem kopīga darba veikšanai tiek nodrošināta brīva un vienlaicīga pieeja programmām un datiem, kas fiziski izvietoti dažādos atmiņas apgabalos.

**311 Copper Distributed Data Interface (CDDI)**
**vara kabeļu dalīto datu saskarne, standarts *CDDI*, *CDDI***
**распределенный интерфейс передачи данных по медным кабелям**

Standarta *FDDI* versija, kurā par datu nesējvidi izmanto vara kabeli. Standarts *CDDI* darbojas līdzīgi *FDDI*, bet tā īstenošana ir lētāka. Tomēr lokālajos tīklos, kuros izmantots standarts *CDDI*, ir samazināts maksimāli pieļaujamais datu pārsūtīšanas ātrums. Līdz ar to šo standartu ieteicams izmantot relatīvi nelielos tīklos, kas izvietoti vienā ēkā.

**312 coprocessor**
**līdzprocesors**
**сопроцессор**

Specializēts procesors, kas papildina centrālā procesora funkcionālās iespējas. Līdzprocesori parasti paplašina komandu kopu, ko var izmantot programmētājs, kā arī nodrošina ātru dažādu matemātisko aprēķinu veikšanu. Dažas centrālā procesora konstrukcijas ļauj izmantot vairākus līdzprocesorus, no kuriem katrs var būt specializēts savu noteiktu funkciju veikšanai. Sk. arī *floating point procesor*, *graphics processor, math coprocessor*.

**313 copy**
**kopēt; kopija**
**копировать; копия**

1. Dublēt datni vai datņu grupu, atstājot oriģinālu esošajā vietā, bet dublikātu novietojot citā direktorijā vai uz cita diska.
2. Teksts, grafiski attēli un zīmējumi, kas apvienoti drukāšanai.

**314 copy protection**
**pretkopēšanas aizsardzība**
**защита от копирования**

Specializēti paņēmieni (piem., apslēptu instrukciju izmantošana, īpaša pieeja diskatmiņai u.c.), kas aizsargā pret nesankcionētu lietojumprogrammu un datņu izmantošanu.

**315 cordless mouse**
**bezvada pele**
**беспроводная мышь**

Pele ar autonomu barošanas avotu, kas nelielā attālumā (līdz 2 m), izmantojot infrasarkanos starus, noraida informāciju personālajam datoram bez vadu palīdzības.

**316 CorelDraw**
**programmatūra *CorelDraw*, *CorelDraw***
**программное обеспечение *CorelDraw***
Firmas *Corel* grafikas programmu pakotne, kas darbojas *Microsoft Windows* vidē. Komplektā ietilpst vektorgrafikas redaktors, fotoattēlu un bitkartētu attēlu apstrādes programma, kā arī trīsdimensiju modelēšanas un trīsdimensiju animāciju veidošanas līdzekļi, kas ērti izmantojami informācijas precizēšanai.

**317 corrective maintenance**
**koriģējošā uzturēšana**
**ремонтное обслуживание**
Uzturēšana, kas nodrošina programmatūras vai aparatūras kļūdu labošanu.

**318 correctness**
**korektums**
**корректность**
1. Pakāpe, kādā sistēma vai tās komponents ir brīvi no specifikācijas izstrādāšanas vai implementācijas procesā pieļautajām kļūdām.
2. Pakāpe, kādā programmatūra, dokumentācija vai citi izmantojamie objekti atbilst specificētajām prasībām.
3. Pakāpe, kādā programmatūra, dokumentācija vai citi izmantojamie objekti atbilst lietotāja vajadzībām.

**319 correctness proof**
**korektuma pierādīšana**
**доказательство корректности**
Sk. *verification*.

**320 corrupted file**
**bojāta datne**
**разрушенный файл**
Datne ar fragmentāriem neatjauninātiem datiem. Datņu bojāšanu var radīt nekvalitatīvi diskatmiņas sektori, kļūmes cieto disku vai diskešu kontrolleru darbībā, kā arī kļūdas programmatūrā.

**321 courtesy copy (CC)**
**cc kopija**
**копия вежливости**
Elektroniskā pasta ziņojums, kas tiek nosūtīts ne tikai tā pamatadresātam, bet arī citām ieinteresētajām personām. Šo personu adreses norādītas ziņojuma galvenē, informējot pamatadresātu par to, kam vēl nosūtīts ziņojums.

**322 CPU**
Sk. *Central Processing Unit*.

**323 CR**
Sk. *carriage return*.

**324 cracker**
**kramplauzis**
**кракер**
Kramplauzis ir tāds urķis, kas cenšas nesankcionēti piekļūt datoru sistēmai vai tīklam. Parasti ar šo terminu apzīmē tīši ļaunprātīgus sistēmu aizsardzības līdzekļu uzlauzējus.

**325 crash**
**avārija**
**авария**
Datora aparatūras vai programmatūras defekts, kas rada neplānotu tā darbības apturi un parasti nav labojams, izmantojot datora iekšējos līdzekļus. Atsevišķos gadījumos, datorus atkārtoti ieslēdzot, avārijas sekas var novērst. Sk. arī *cold boot*.

**326 critical software**
**kritiskā programmatūra**
**критическое программное обеспечение**

Programmatūra, kuras kļūme var būtiski pazemināt tās nebīstamības pakāpi un radīt ievērojamus finansiālus vai sociālus zaudējumus.

**327 cropping**
**apcirpšana**
**редактирование изображения с удалением его частей**

Grafiska attēla rediģēšana, kurā apgriež attēla malas vai atmet nevajadzīgās daļas, lai to varētu ievietot atvēlētajā displeja ekrāna laukumā.

**328 CRT display**

Sk. *cathod-ray tube display.*

**329 cryptography**
**kriptogrāfija**
**криптография**

Zinātnes nozare, kas pēta ziņojumu šifrēšanas metodes un līdzekļus. Ar kriptogrāfijas metodēm aizsargāto ziņojumu var dešifrēt tikai tas, kam ir šifra atslēga. Eksistē daudz dažādu paņēmienu, kas sākotnējo ziņojuma tekstu pārveido šifrētā tekstā, ko iespējams izlasīt, tikai izmantojot šifra atslēgu vai t.s. kriptoanalīzes metodes. Pēdējās ir speciālu šifra apstrādes paņēmienu kopums, kas ļauj atjaunot sākotnējo informāciju personai, kura nezina šifra atslēgu vai tās izmantošanas algoritmu.

**330 CSMA/CA**

Sk. *Carrier Sense Multiple Access with Collision Avoidance.*

**331 CSMA/CD**

Sk. *Carrier Sense Multiple Access with Collision Detection.*

**332 Ctrl key**

Sk. *Control key.*

**333 CUI**

Sk. *Character-based User Interface.*

**334 current directory**
**aktuālais direktorijs**
**текущий каталог**

Direktorijs, ar kuru operētājsistēma vai lietojumprogramma strādā. Komandu uzvedne parasti norāda aktuālo direktoriju un diskdzini.

**335 current drive**
**aktuālais diskdzinis**
**текущий дисковод**

Diskdzinis, ko operētājsistēma izmanto savai darbībai.

**336 cursor**
**kursors**
**курсор**

1. Pārvietojams simbols, kas displeja ekrānā izpilda kontaktpunkta lomu lietotāja darbībā ar datiem. Teksta apstrādes sistēmās kursors parasti ir mirgojošs taisnstūris vai pasvītrojums. Grafikas sistēmās to bieži sauc arī par rādītāju un tam var būt dažāda forma (bultiņa, kvadrāts, otiņa u.c.), ko tas var mainīt, pārvietojoties pa displeja ekrānu. Sk. arī *mouse pointer.*

2. Zīmulim vai pildspalvai līdzīga ierīce, ko izmanto grafikas planšetēs. Pārvietojot šādu ierīci pa planšeti, atbilstoši pārvietojas kursors displeja ekrānā.

**337 cursor keys**
**kursortaustiņi**
**клавиши управления курсором**
Taustiņi, ar kuriem pa displeja ekrānu pārvieto kursoru. Pie kursortaustiņiem pieskaitāmi bulttaustiņi un paplašinātās tastatūras taustiņi: sākumvietas taustiņš, beigvietas taustiņš, tabulēšanas taustiņš, augšupšķiršanas taustiņš un lejupšķiršanas taustiņš.

**338 cursor-movement keys**
**kursorvirzes taustiņi**
**клавиши перемещения курсора**
Sk. *Arrow keys*.

**339 cut**
**izgriezt**
**вырезать**
Dokumenta vai tā daļas (piem., teksta, grafikas) pārvietošana starpliktuvē, lai to varētu izmantot šī paša vai cita dokumenta rediģēšanas vai sagatavošanas procesā.

**340 cut and paste**
**izgriezt un ielīmēt**
**вырезать и вставить**
Dokumenta rediģēšanas paņēmiens, kas dod iespēju kādu iezīmētu dokumenta daļu (piem., tekstu, grafiku) izgriezt un ievietot kādā šī paša vai cita dokumenta noteiktā vietā.

**341 cyberspace**
**kibertelpa**
**киберпространство**
Terminu izmanto, lai apzīmētu datoru tīklā (t.sk. tīklā *Internet*) veidoto diskrēto pasaules modeli (virtuālo realitāti). Terminu radījis V. Gibsons savā romānā *"Neuromancer"*.

**342 cycle stealing**
**cikla pārķere**
**выделение цикла**
Tāda centrālā procesora darba cikla izmantošana, kas ļauj periodiski pārtvert atsevišķus pamatatmiņas darba ciklus un nodrošināt vienlaicīgu vairāku datu apstrādes un perifērijas ierīču operāciju veikšanu vai zināmu to izpildes pārklāšanos.

## D

**343 daemon**
**dēmons**
**демон**
Fona programma, kas vajadzības gadījumā tiek izmantota datoros, kuri darbojas *UNIX* vai kādas citas operētājsistēmas vidē. Dēmons veic tādus pakalpojumus kā, piem., elektroniskā pasta ziņojumu maršrutēšanu, un tas parasti darbojas bez lietotāja iejaukšanās.

**344 DAT**
Sk. *digital audio tape*.

**345 data**
**dati**
**данные**
Formalizētā veidā attēlota tekstuāla, skaitliska, grafiska, video- un audioinformācija, kuru lietotājs vai datu apstrādes ierīces var interpretēt, apstrādāt vai pārsūtīt.

**346 data array**
**datu masīvs**
**массив данных**
Sk. *array*.

347 **data bank**
**datu banka**
**банк данных**
Automatizēta datu krātuve (bibliotēku kopums) ar lietotājiem vajadzīgiem attiecīgās nozares datiem. Datu bankā dati var tikt organizēti gan kā datu bāze, gan kā viena vai vairākas datnes.

348 **data block**
**datu bloks**
**блок данных**
Datu kopums, ko kā vienotu veselumu pārsūta starp dažādiem datora funkcionālajiem blokiem vai datora tīkla komponentiem.

349 **data communication**
**datu apmaiņa**
**обмен данными**
Informācijas pārsūtīšana no viena datora otram, izmantojot kabeļus vai telefona līnijas un modemus. Datu apmaiņa notiek atbilstoši izvēlētajam protokolam.

350 **data compression**
**datu saspiešana**
**сжатие данных**
1. Lielāka informācijas daudzuma ierakstīšana noteiktas ietilpības atmiņā, izmaintojot dažādus datu organizēšanas vai kodēšanas paņēmienus.
2. Datu pārveidošana kompaktā formā bez informācijas zaudējumiem. To izmanto datu pārraides sistēmās, lai paaugstinātu šo sistēmu efektivitāti.

351 **data definition language (DDL)**
**datu definēšanas valoda,**
**valoda *DDL***
**язык определения данных**
Valoda, kas parasti ir datu bāzes pārvaldības sistēmas sastāvdaļa un ko izmanto, lai aprakstītu visus datu bāzes atribūtus un īpašības (piem., ierakstu izvietojumu, atslēgu laukus, datņu atrašanās vietas u.c.).

352 **data deletion**
**datu dzēšana**
**удаление данных**
Datu bāzes pārvaldības sistēmā — datu izņemšana no datu bāzes atbilstoši noteiktiem kritērijiem. Datu dzēšanu var veikt automātiski vai manuāli. Sk. arī *data manipulation*.

353 **data dictionary**
**datu vārdnīca**
**словарь данных**
Datu bāzes pārvaldības sistēmā — tabula, kurā ir informācija par visiem datu bāzes elementiem. Datu vārdnīcā ir ziņas par ierakstu un datņu tipiem, indeksi un cita informācija par datu bāzes izmantošanu.

354 **data entry**
**datu ieeja**
**вход данных**
Tieša datu ievade datorā, izmantojot tastatūru, skeneri vai citas ierīces.

355 **data field**
**datu lauks**
**поле данных**
Vieta, kas ierakstā rezervēta noteiktas informācijas glabāšanai. Relāciju datu bāzēs, kur dati tiek attēloti tabulu kopas veidā, ieraksti tabulās veido rindiņas, bet datu lauki — kolonnas.

356 **data flow**
**datu plūsma**
**поток данных**

1. Datu kopums, ko noteiktā secībā no datu avota pārraida datu saņēmējam.
2. Secība, kādā datorā datus pārraida, izmanto un pārveido programmas izpildes gaitā.

357 **data independence**
**datu neatkarība**
**независимость данных**
Datu bāzes pārvaldības sistēmā — datu atdalīšana no programmas, kas tos izmanto. Programmai, kas izmanto datus, nav nepieciešama informācija par to, kā dati izvietoti datu bāzē. Tādējādi ir iespējams veikt izmaiņas datu bāzē, minimāli mainot lietojumprogrammas.

358 **data integrity**
**datu integritāte**
**целостность данных**
Datu bāzes pārvaldības sistēmā — datu pareizība, iekšējais nepretrunīgums, kā arī informācijas pilnīgums datu bāzēs, saistītās datnēs vai tabulās. Datu integritāti svarīgi saglabāt pēc datu atjaunināšanas vai dzēšanas.

359 **data manipulation**
**datu manipulēšana**
**манипулирование данными**
Datu bāzes parvaldības sistēmā — tādu datu bāzu manipulēšanas pamatoperāciju kā datu dzēšana, datu iespraušana, datu modificēšana un datu izgūšana izpilde. Sk. arī *data manipulation language*.

360 **data manipulation language (DML)**
**datu manipulēšanas valoda**
**язык манипулирования данными**
Valoda, kas parasti ir datu bāzu pārvaldības sistēmas daļa un kas atvieglo datu manipulēšanas operāciju izpildi.

361 **data medium**
**datu vide**
**носитель данных**
Fizikālā vide (materiāls), kurā tiek ierakstīti dati.

362 **data model**
**datu modelis**
**модель данных**
Datu loģiskās struktūras organizācija datu bāzē. Datu bāzes pārvaldības sistēma parasti uztur tikai vienu datu modeli. Sk. arī *data manipulation*, *data manipulation language*.

363 **data modification**
**datu modificēšana**
**модификация данных**
Datu bāzes pārvaldības sistēmā — jebkura transakcija, kas atbilstoši pieņemtajiem kritērijiem atjauno vienu vai vairākus ierakstus datu bāzē. Sk. arī *data manipulation*.

364 **data processing**
**datu apstrāde**
**обработка данных**
Datu vākšana, glabāšana, modificēšana, izguve un citas operācijas ar datiem.

365 **data projector**
**datu projektors**
**проектор данных**
Videoierīce, kas projicē datora izvaddatus (t.sk. krāsu attēlus) uz attāla lielu izmēru ekrāna.

366 **data record**
**datu ieraksts**
**запись данных**
Sk. *record*.

367 **data reduction**
**datu reducēšana**
**уплотнение данных**
Sākotnējo datu (jēldatu) pārveidošana kompaktākā, strukturētākā vai citādā labāk izmantojamā formā, lietojot filtrēšanu, mērogošanu, sakārtošanu vai citas metodes. Datu reducēšanu parasti veic pirms to nodošanas apstrādei lietojumprogrammām.

368 **data redundancy**
**datu redundance**
**избыточность данных**
1. Tādu dublējošu vai papildu datu esība informacionālajos masīvos, kuru izslēgšana no masīva neiespaido tā adekvātumu risināmajai problēmai.
2. Vienu un to pašu datu atkārtošanās divos vai vairākos datu ierakstos. No šādas datu redundances vēlams pēc iespējas izvairīties, jo tā var radīt vienkāršas kļūdas (piem., burtu atkārtošanos). Relāciju datu bāzes pārvaldības programmas parasti pieļauj samazināt datu redundanci.

369 **data retrieval**
**datu izgūšana**
**выборка данных**
Datu atgūšana, meklējot tos un nolasot no datnes, atmiņas, ārējās iekārtas vai datu bāzes. Sk. arī *data manipulation*.

370 **data structure**
**datu struktūra**
**структура данных**
Datu elementu savstarpējās fiziskās un loģiskās attiecības, kas izveidotas, lai veiktu īpašas datu apstrādes funkcijas.

371 **data table**
**datu tabula**
**таблица данных**
Datu bāzes pārvaldības sistēmā — informācijas attēlojums displeja ekrānā divdimensiju masīva (tabulas) veidā. Parasti datu bāzu pārvaldības programmas izspīdina datu tabulas, izpildot kārtošanas vai vaicājumu operācijas. Sk. arī *data-entry form*.

372 **data type**
**datu tips**
**тип данных**
Datu paveids. Raksturīgākie datu tipi ir skaitliskie dati, burtciparu (rakstzīmju) dati, loģiskie (*patiess/aplams*) dati un datumi. Programmēšanas valodas parasti nodrošina dažādu datu tipu veidošanu. Datu bāzu pārvaldības sistēmās šo terminu izmanto, lai klasificētu datu laukus, kuri nosaka, kādus datus var ievadīt datu bāzē.

373 **data-entry form**
**datu ievadforma**
**формуляр ввода данных**
Datu bāzes pārvaldības sistēmā — displeja ekrānā attēlojamo datu forma, kas atvieglo datu ievadīšanu un rediģēšanu, jo vienlaicīgi ekrānā tiek izspīdināts tikai viens ieraksts.

374 **database**
**datu bāze**
**база данных**
Savstarpēji saistītu informacionālu objektu tematisks kopums, kas ar speciālas pārvaldības sistēmas starpniecību organizēts tā, lai nodrošinātu ērtu informācijas izguvi, izdarītu tās atlasi un kārtošanu. Informācija datu bāzē parasti ir sadalīta ierakstos (tabulās), no kuriem katram var būt viens vai vairāki lauki (kolonnas). Sk. arī *database management system*.

375 **database administrator**
**datu bāzes administrators**
**администратор базы данных**
Persona vai personu grupa, kas atbild par datu bāzes uzturēšanu, ekspluatāciju, attīstību, drošību un integritāti.

376 **database key**
**datu bāzes atslēga**
**ключ базы данных**
Atslēga, ko piešķir datu bāzes pārvaldības sistēma un kas viennozīmīgi identificē datu bāzes ierakstu.

377 **database management system**
**datu bāzes pārvaldības sistēma**
**система управления базой данных**
Lietojumprogrammatūra, kas organizē datus datu bāzē, nodrošinot to uzglabāšanu, izguvi, drošību un integritāti. Datu bāzes pārvaldības sistēma parasti veic izdrukājamo pārskatu formatizēšanu, kā arī eksportu uz un importu no citām lietojumprogrammām, izmantojot datņu standartformātus. Sk. arī *data model*, *data manipulation*.

378 **database server**
**datu bāzes serveris**
**сервер базы данных**
Lokālā tīkla dators, kas paredzēts koplietošanas datu bāzes uzglabāšanai un datu izguvei no tās.

379 **daughter board**
**meitasplate**
**дочерняя плата**
Drukātās shēmas plate, ko ievieto mātesplates slotā, lai paplašinātu datora funkcionālās iespējas.

380 **dBase**
**datu bāzes pārvaldības sistēma *dBase*, sistēma *dBase*, *dBase***
**система управления базой данных *dBase***
Firmas *Ashton-Tate Corporation* izveidota datu bāzes pārvaldības sistēma. Tā ir pirmā datu bāzes pārvaldības sistēma, kas izveidota *IBM* saderīgiem personālajiem datoriem. *dBase* programmēšanas valoda un datņu formāti jau ir kļuvuši par rūpnieciskiem standartiem. Sistēmā *dBase* ir interpretējoša, valodai *Pascal* līdzīga valoda un arī komandas interaktīvas vadības sistēmas lietošanai. 1991. g. firmu *Ashton-Tate Corporation* nopirka firma *Borland International Inc.*, kas izveidojusi šīs sistēmas nākošo pakāpi operētājsistēmas *Microsoft Windows* videi (*dBase for Windows*).

381 **DD disk**
Sk. *double density disk*.

382 **DDE**
Sk. *Dynamic Data Exchange*.

383 **DDL**
Sk. *data definition language*.

384 **deadlock**
**strupceļš**
**тупик**
Situācija datu apstrādes sistēmā, kad divi procesi, kas pieprasa vienus un tos pašus resursus, bloķē viens otru.

385 **debugging**
**atkļūdošana**
**отладка**
Procedūra programmas sastādīšanas vai shēmas izstrādāšanas gaitā pieļauto

kļūdu atrašanai, lokalizēšanai un novēršanai, ko parasti veic ar datora palīdzību. Sk. arī *bug*.

### 386 DEC

Sk. *Digital Equipment Corporation*.

### 387 decryption
### atšifrēšana
### дешифрование

Šifrēta ziņojuma apstrāde, ko veic tā saņēmējs, lai atklātu ziņojuma sākotnējo saturu. Sk. arī *ciphertext*, *cryptography*, *encryption*.

### 388 default
### noklusējums
### умолчание

Aparatūras un programmatūras darbība vai standartuzstādījums gadījumos, kad lietotājs nav uzdevis nekādu alternatīvu. Tā, piem., ja tekstu apstrādes programmās lietotājs nav uzdevis malu izmērus, lappuses garumu vai citus parametrus, tad sistēma pati piešķir tiem noteiktas standartvērtības.

### 389 default button
### noklusējuma poga
### кнопка, использованная по умолчанию

Izgaismota vai speciāli apzīmēta poga, ko grafiskajā lietotāja saskarnē sistēma automātiski izvēlas kā lietotāja visiespējamāko izvēli. Sk. arī *Cancel button*, *default choice*, *OK button*, *pushbutton*.

### 390 default choice
### noklusējuma izvēle
### выбор по умолчанию

Izvēle, ko programma veic automātiski. Parasti noklusētā izvēle ir visiespējamākā izvēle, ko lietotājs būtu izdarījis kādā izvēlnē vai sarakstā.

### 391 default drive
### noklusējuma diskdzinis
### дисковод по умолчанию

Diskdzinis, ko operētājsistēma izmanto, lai no tā nolasītu vai tajā ierakstītu informāciju, ja lietotājs nav norādījis citu diskdzini.

### 392 default font
### noklusējuma fonts
### фонт по умолчанию

Burtveidols vai rakstzīmes fonts, ko sistēma izmanto teksta drukāšanai, ja lietotājs nav devis citus norādījumus.

### 393 default printer
### noklusējuma printeris
### принтер по умолчанию

Printeris, ko lietojumprogramma izmanto automātiski tajā gadījumā, ja lietotājs nav devis norādījumus par cita printera lietošanu.

### 394 defragmentation
### defragmentēšana
### дефрагментация

Datņu pārrakstīšana disketē vai cietajā diskā tā, lai visas datnes daļas būtu ierakstītas blakus esošajos diska sektoros. Defragmentēšana būtiski samazina informācijas izguves laiku.

### 395 Del key

Sk. *Delete key*.

### 396 Delete key
### dzēšanas taustiņš
### клавиша стирания

Taustiņš, kas apvienots ar decimāldalī-

tāja taustiņu un izvietots firmas *IBM* un ar to saderīgas tastatūras cipartastatūras daļā kā arī paplašinātās tastatūras rediģēšanas taustiņu blokā starp tastatūras pamatdaļu un cipartastatūru. Dzēšanas taustiņam dažādās programmās un lietojumos var būt atšķirīgas funkcijas. Visbiežāk tas dzēš nākamo rakstzīmi aiz kursora, bet dažās programmās tas var dzēst arī iezīmēto tekstu, attēlu vai citu objektu.

397 **demand paging**
**pieprasījumlapošana**
**подлистывание по запросу**

Atmiņas iedalīšanas tehnika, kurā lappuses no palīgatmiņas tiek pārsūtītas uz pamatatmiņu tikai tad, kad šīs lappuses ir vajadzīgas uzdevuma izpildei. Sk. arī *anticipatory paging*.

398 **descriptor**
**deskriptors**
**дескриптор**

Valodas vārds vai vārdu grupa, ko izmanto informācijas klasificēšanai vai indeksēšanai. Datu bāzes pārvaldības sistēmās — vārds, ko izmanto ierakstu klasificēšanai un kas dod iespēju iegūt visu ierakstu grupu, kura satur šo vārdu. Dažkārt deskriptoru sauc arī par atslēgas vārdu.

399 **desk accessories**
**darbvirsmas piederumi**
**принадлежности рабочей поверхности**

*Macintosh* saimes datoros vai logošanas vides programmās izmantojamas nelielas programmas, kas darbojas, piem., kā elektronisks ekvivalents pulkstenim, kalendāram, kalkulatoram vai kādam citam objektam, kurš parasti atrodams uz galda.

400 **desktop**
**darbvirsma**
**рабочая поверхность**

Displeja ekrāna apgabals, kas imitē galda virsmu. Darbvirsma dod iespēju lietotājam pārvietot objektus, sākt un beigt uzdevumu izpildi tāpat kā uz reālas galda virsmas, tādējādi atvieglojot lietotāja darbu ar datoru.

401 ***desktop computer***
**galddators**
**настольный компьютер**

Maza izmēra personālais dators, kam pamatatmiņa un diskatmiņa ir pietiekami ietilpīga, lai risinātu dažāda veida lietojumuzdevumus, un kas ir piemērots novietošanai uz galda. Sk. arī *personal computer*, *microcomputer*.

402 **desktop presentation**
**datorpriekšstāde,**
**datorprezentācija**
**настольное представление,**
**настольная презентация**

Pasniegšanas grafikas programmās esošās slīdrādes iespēju izmantošana, lai displeja ekrānā izspīdinātu diagrammas vai citu ilustratīvu materiālu, kas atrodas datorā.

403 **desktop publishing**
**datorizdevniecība**
**настольные издательские средства**

Datora un specializētas programmatūras izmantošana, lai, apvienojot tekstu un grafiku, izveidotu materiālu, ko var tiražēt.

404 **destination**
**adresāts, saņēmējs**
**место назначения**
Ieraksts, datne, dokuments vai disks, uz kuru informāciju pārsūta vai kurā to pārkopē. Sk. arī *destination address*, *source*.

405 **destination address**
**saņēmēja adrese**
**адрес назначения**
Ierīces vai atmiņas apgabala adrese, uz kuru tiek pārsūtīti dati. Sk. arī *destination*, *source*, *source address*.

406 **destructive read**
**destruktīvā lasīšana**
**считывание с разрушением**
Datu nolasīšanas operācija, kuru izpildot dati tiek pārsūtīti datora procesoram, bet to kopija datora atmiņā tiek mainīta vai iznīcināta.

407 **developer's toolkit**
**izstrādātāja aprīkojums**
**набор инструментальных средств разработчика**
Palīgprogrammu (rutīnu) kopa, kas lietojumprogrammu izstrādātājam atvieglo programmu veidošanu, ievērojot konkrētās to darbības vides īpatnības (piem., grafisko lietotāja saskarni, operētājsistēmu, datu bāzu pārvaldības sistēmu u.c.). Sk. arī *multimedia developer's kit*, *toolkit*.

408 **development system**
**izstrādes sistēma**
**система разработки**
1. Programmēšanas valoda un ar to saistītās palīgprogrammas (kompilatori, teksta redaktori, atkļūdotāji u.c.), kas atvieglo programmu sastādīšanu.

2. Dators ar specializētu programmatūru, kas atvieglo noteikta lietojuma datorsistēmas izveidošanu.

409 **device**
**ierīce**
**устройство**
Jebkurš aparatūras komponents, to apvienojums vai ārējā iekārta, ko izmanto, piem., datu saņemšanai, apstrādei, uzglabāšanai vai nosūtīšanai. Dažu ierīču darbināšanai nepieciešama speciāla programmatūra — ierīču draiveri. Sk. arī *compound device*, *input device*, *output device*, *pointing device*.

410 **device driver**
**ierīces draiveris**
**драйвер устройства**
Programma, kas nodrošina operētājsistēmas sadarbību ar kādu perifērijas ierīci, sniedzot oprētājsistēmai informāciju par tās raksturojumiem (piem., displeja ekrāna izšķirtspēju). Ja datoram tiek pieslēgtas jaunas ārējās iekārtas, tad jāizmanto jauni draiveri.

411 **device name**
**ierīces vārds**
**имя устройства**
Nosaukums, ar kuru operētājsistēma identificē ierīci, t.i., nosaka tās fizisko adresi.

412 **DGIS**
Sk. *Direct Graphics Interface Specification*.

413 **Dhrystone**
**etalonprogramma *Dhrystone***
**эталонная тестовая программа *Dhrystone***

Etalonprogramma, ko izmanto, lai noteiktu un salīdzinātu dažādu datoru veiktspēju, tiem izpildot vispārēja rakstura instrukcijas. Veiktspēja tiek raksturota ar šīs programmas izpildes reižu skaitu sekundē. Sk. arī *Whetstone*.

**414 diagnostic message**
**diagnostiskais ziņojums**
**диагностическое сообщение**

Ziņojums, ko izstrādā diagnostikas programmas vai utilitprogrammas un kas satur informāciju par pārbaudāmās programmas vai ierīces disfunkciju raksturu un to atrašanās vietu.

**415 dial-up access**
**iezvanpieeja**
**доступ набором номера**

Iespēja nodibināt savienojumu ar citu datoru vai datoru tīklu (piem., tīklu *Internet*), izmantojot modemu. Tīkla *Internet* pakalpojumu sniedzējs par noteiktu atlīdzību nodrošina iezvanpieeju tīkla *Internet* resursiem.

**416 dial-up account**
**iezvankonts**
**счет набора номера**

Tīkla *Internet* konta pamatveids, kas lietotājam, izmantojot modemu, dod iespēju par attiecīgu samaksu saņemt tīkla *Internet* pakalpojumus.

**417 dial-up line**
**iezvanlīnija**
**коммутируемая линия**

Komutējama sakaru līnija, kas paredzēta īslaicīgai divu datoru sadarbībai, izmantojot vispārējās lietošanas telefona kanālus. Datoru sadarbība līdzīgi sakaru nodibināšanai starp telefona abonentiem tiek organizēta ar automātiskās telefonu centrāles starpniecību.

**418 dialer program**
**zvanītājprogramma**
**программа подключения с вызовом по номеру**

Programma, kas "uzgriež" tīkla *Internet* pakalpojumu piegādātāja numuru un nodibina ar to savienojumu. Atšķirībā no komunikāciju programmām, kas pārveido lietotāja datoru attālā terminālī, zvanītājprogrammas nodibinātais savienojums pilnībā integrē lietotāja datoru tīklā *Internet*. Sk. arī *Point-to-Point Protocol, Serial Line Internet Protocol*.

**419 dialog box**
**dialoglodziņš**
**диалоговое окно**

Lodziņš displeja ekrānā, ar kura starpniecību lietotājs nodod pieprasīto informāciju sistēmai vai lietojumprogrammai, lai nodrošinātu tās izpildes procesa turpināšanu. Sk. arī *message box, primary window, secondary window*.

**420 digital audio tape (DAT)**
**ciparu audiolente**
**цифровая аудиолента**

Magnētiskā lente, kas nodrošina ciparsignālu formā pārveidotas skaņas ierakstu, kura atskaņošanas kvalitāte ir salīdzināma ar reproducēšanas kvalitāti, ko nodrošina kompaktdiski.

**421 digital camera**
**ciparkamera**
**цифровая камера**

Videokamera, kas ar speciāla sensora starpniecību katram attēla pikselim atbilstošo gaismas intensitāti pārveido ciparsignālā.

kodā, ko var ierakstīt kameras brīvpieejas atmiņā vai nelielā disketē. Ciparkamera krāsu attēlu ciparkamera veido, nosakot katras atsevišķās krāsas intensitāti un piešķirot katram sarkanās, zaļās un zilās krāsas pikselim noteiktu vērtību.

422 **digital cassete**
**ciparkasete**
**цифровая кассета**
Augstas kvalitātes magnētiskā lente, kas ievietota standartizmēra kasetē ar ierakstaizsardzības un standartformāta nodrošināšanas iespējām.

423 **digital control monitor**
**ciparvadības monitors**
**монитор цифрового управления**
Monitors, kura attēla parametrus (lielumu, pozīciju utt.) var uzdot un to izmaiņas vadīt ar ciparkodu.

424 **Digital Equipment Corporation (DEC)**
**firma *Digital Equipment Corporation*, firma *DEC*, *DEC***
**фирма *DEC***
Viena no lielākajām datoru un citu elektronisko iekārtu ražotājām firmām ASV. Firma *DEC* bija pirmā, kas jau 1965. g. uzsāka minidatoru *PDP-8* ražošanu. Izmantojot šī minidatora arhitektūru, vēlāk firma izstrādāja tekstu apstrādes datoru saimi *DECmate*. Astoņdesmito gadu sākumā izstrādātais personālais dators *VAXmate* jau bija daļēji saderīgs ar *IBM PC*. Turpmāk šīs firmas personālie datori (*DECpc*, *Celebris*, *Venturis*) veidoti, izmantojot mikroprocesorus *Intel 80486* vai *Intel Pentium* un tie ir pilnīgi saderīgi ar *IBM PC*. 1992. gadā firma *DEC* izstrādāja arhitektūru *Alpha AXP* un uz tās bāzes izveidoja darbstacijas *DEC 3000* ar lielu veiktspēju un plašām lietojuma iespējām.

425 **digital monitor**
**ciparmonitors**
**цифровой монитор**
Monitors, kas, lai izgaismotu ekrānu, no datora saņemtos ciparsignālus pārveido analogsignālos. Sk. arī *analog monitor*.

426 **digital optical recording**
**optiskā ciparierakste**
**оптическая цифровая запись**
Signālu ierakstīšana binārā formā, veidojot nelielus iedobumus optisko disku vai kompaktdisku virsmā. Ieraksta nolasīšanai izmanto lāzera staru.

427 **digital speech**
**ciparruna**
**цифровая речь**
Sk. *speech synthesis*.

428 **Digital Video-Interactive (DV-I)**
**interaktīvā ciparu video-sistēma, sistēma *DV-I*, *DV-I***
**интерактивная цифровая видеосистема**
Aparatūras un programmatūras sistēma, kas veic diskrētās video- un audioinformācijas saspiešanu tās tālākai izmantošanai personālajos datoros.

429 **digitized photography**
**ciparfotogrāfija**
**оцифрованная фотография**
Attēls vai fotogrāfija, kas tiek skenēts, lai iegūtu analogsignālu, kuru savukārt pārveido ciparsignāla formā, lai attēlu uzglabātu datorā vai izspīdinātu displeja ekrānā.

**430 digitizer**
**ciparotājs**
**цифровой преобразователь**
Ievadierīce, ar kuru operators veic grafiskās informācijas pārveidošanu cipardatos un to ievadīšanu datorā.

**431 digitizing pad**
**ciparošanas paliktnis**
**планшетный цифровой преобразователь**
Sk. *digitizing tablet.*

**432 digitizing tablet**
**ciparošanas planšete**
**планшет для цифрового ввода**
Plastmasas planšete ar jutīgu virsmu, kas pārveido speciālas rādītājierīces (irbuļa vai kursora) pozīciju ciparsignālā, lai ievadītu to datorā un attēlotu uz displeja ekrāna. Sk. arī *digitizer.*

**433 dingbat**
**dekorzīme**
**декоративный знак**
Nelieli grafiski elementi, ko izmanto dokumenta dekoratīvai noformēšanai, piem., zvaigznītes, bultiņas, ģeometriskas figūras.

**434 DIP**
Sk. *dual in-line package.*

**435 Direct Graphics Interface Specification (DGIS)**
**tiešās grafiskās saskarnes specifikācija, standarts *DGIS*, *DGIS***
**интерфейс *DGIS***
Firmas *Graphics Software Systems* izstrādāts saskarnes standarts, kas, izmantojot lasāmatmiņu, parasti implementēts kā programmatūra videoadapteros un kas dod iespēju programmai attēlot grafiku displeja ekrānā.

**436 directory**
**direktorijs**
***директория***
Datora diskatmiņā ierakstīta datņu vārdu un apakšdirektoriju tabula. Direktorijs norāda saistību starp šīm datu struktūrām un to izvietojumu atmiņā.

**437 directory listing**
**direktorija saraksts**
**список каталога**
Direktorijā ietverto datņu un apakšdirektoriju saraksts.

**438 directory tree**
**direktoriju koks**
**дерево каталогов**
Grafisks diska satura attēlojums, kas rāda direktoriju un apakšdirektoriju savstarpējās attiecības un to hierarhiju.

**439 disabling**
**atspējošana**
**отмена**
Īslaicīga aparatūras vai programmatūras funkcionēšanas pārtraukšana, lai padarītu tās izmantošanu neiespējamu. Sk. arī *enabling.*

**440 disk**
**disks**
***диск***
Datu vide, kas izveidota kā vienas vai vairāku apaļu plašu komplekts, kas pārklātas ar tāda materiāla kārtiņu, kurā var ierakstīt un no kuras var nolasīt datus. Izšķir magnētiskos, magnēto-optiskos un

optiskos diskus. Sk. arī *floppy disk*, *hard disk*.

**441 disk cache**
**diska kešatmiņa**
**дисковая кэш-память**

Datora atmiņas apgabals, kurā īslaicīgi uzglabā bieži lietojamus datus. Diska kešatmiņa izpilda starpnieka funkcijas starp lietojumprogrammu un diskatmiņu. Speciāla programma vispirms pārbauda, vai nepieciešamie dati ir diska kešatmiņā, un tikai tādā gadījumā, ja tur šo datu nav, griežas pie cietā diska. Lai paātrinātu informācijas izguvi, diska kešatmiņu izmanto arī globālā tīmekļa *WWW* programmas, lai tur uzglabātu bieži lietojamus dokumentus.

**442 disk cartridge**
**diska kasetne**
**кассета для магнитного диска**

Plastmasas vai metāliskā kasetnē iebūvēts cietais disks, ko var ielikt vai izņemt no diskdziņa līdzīgi disketei. Šādai kasetnei ir daudz lielāka ietilpība un piekļūšanas ātrums nekā lokanajam diskam.

**443 disk drive**
**diskdzinis**
**дисковод**

Ārējā ierīce, kas nodrošina datu ierakstīšanu un nolasīšanu no diskiem. Diskdziņa sastāvā ietilpst lasīšanas/rakstīšanas galviņas, nepieciešamās elektroniskās shēmas un diska piedziņas mehānisms.

**444 disk duplicator**
**diska dublētājs**
**копировщик диска**

Ierīce, kas formatē un izveido lokano disku kopijas programmatūras izplatīšanai.

**445 disk jacket**
**disketes apvalks**
**кожух диска**

No disketes neatdalāms plastmasas futlāris, kas netraucē informācijas ierakstīšanu un nolasīšanu.

**446 disk operating system (DOS)**
**operētājsistēma *DOS*, *DOS***
**дисковая операционная система, *DOS***

1. Kopīgs apzīmējums operētājsistēmām, kas sistēmas starta vai restarta gadījumā tiek ielādētas no diska.
2. Operētājsistēma firmas *IBM* personālo datoru saimēm: *IBM PC* un *IBM PS/2*. Šo operētājsistēmu parasti sauc par *PC-DOS*, lai atšķirtu to no operētājsistēmas *MS-DOS*, kas ir firmas *Microsoft* produkts.

**447 disk partition**
**diska sadaļa**
**сегмент дисковой памяти**

Fiziskā diska loģiskā daļa. Fiziskajam diskam var būt viena vai vairākas diska sadaļas. Katrai no tām ir atšķirīgs diskdziņa nosaukums.

**448 diskette**

Sk. *floppy disk*.

**449 diskless workstation**
**bezdiska darbstacija**
**бездисковая рабочая станция**

Dators bez diskatmiņas, kas strādā datoru tīklā un saņem datus un programmas no tīkla servera.

**450 display**
**displejs**
**дисплей**

Ievadizvades ierīce, kas ar ekrāna starpniecību dod iespēju vizuālā formā ievadīt datorā un izvadīt no tā tekstu un dažāda veida grafisku informāciju. Sk. arī *cathod ray tube display, light-emitting diode display, liquid crystal display, monitor.*

**451 Display PostScript**
**valoda *Display PostScript***
**язык *Display PostScript***

Valodas *PostScript* paplašinājums, kas interpretē valodas *PostScript* instrukcijas un displeja ekrānā izveido tieši tādus grafikas un teksta elementus, kādus lietotājs tos ieraudzīs izdrukātajā dokumentā.

**452 display power management system (DPMS)**
**displeja barošanas pārvaldības sistēma, sistēma *DPMS***
**система управления электропитанием дисплея**

Sistēma, kurā speciāli aprīkots videoadapteris nodod monitoram instrukcijas par elektroenerģijas taupīšanu. Parasti tiek paredzēti vairāki elektroenerģijas taupīšanas līmeņi.

**453 display screen**
**displeja ekrāns**
**экран дисплея**

Sk. *monitor.*

**454 distributed database**
**dalītā datu bāze**
**распределенная база данных**

Datu bāze, kas sadalīta pa vairākām datortīkla stacijām un ir pieejama kopīgai izmantošanai dažādiem lietotājiem. Lai gan dalītā datu bāze ir ģeogrāfiski izvietota dažādās vietās, tās pārvaldības sistēma to apstrādā kā vienotu veselumu. Izmantojot speciālas datu izguves un pārsūtīšanas metodes, datu bāzes dalīšana var būtiski uzlabot tās veiktspēju.

**455 Distributed Queue Dual Bus (DQDB)**
**dalītas rindas dubultkopne, kopne *DQDB***
**двойная шина с распределенной очередью, шина *DQDB***

Standartā *IEEE 802.6* izmantotā pilsēttīklu izveides topoloģija, kas paredz informācijas pārraidi pa divām paralēlām kopnēm, kuras darbojas neatkarīgi viena no otras. Šāda topoloģija dod iespēju palielināt tīkla veiktspēju un defekttoleranci. Sk. arī *IEEE 802 Standards.*

**456 dithering**
**tonēšana**
**размывание**

Datorgrafikā — papildkrāsu un papildtoņu veidošana no esošās krāsu paletes. Vienkrāsas displejiem pelēkuma toņus veido, mainot punktu šablonus un to blīvumu. Krāsu displejiem krāsas un šabloni tiek veidoti, apvienojot dažādu eksistējošo krāsu punktus. Tonēšanu lieto attēlu foniem un dažādiem aizpildījumiem.

**457 DML**

Sk. *data manipulation language.*

**458 document format**
**dokumenta formāts**
**формат документа**

Dokumenta ārējais veidols, kas tam piešķirts formatēšanas procesā un ko nosaka

tādi tā elementi kā, piem., malu platums, galvenes un parindes forma, lappušu numuru izvietojums, teksta dalījums slejās u.c.

**459 document image processing**
**dokumenta attēla apstrāde**
**обработка изображения документа**

Dokumentu apstrāde, izmantojot to attēlus, ko parasti ievada datorā ar skeneri. Ar speciālas programmatūras starpniecību šie dokumenti tiek ievietoti dokumentu datu bāzē, kur tos ērti sameklēt.

**460 document window**
**dokumenta logs**
**окно документа**

Grafiskajā lietotāju saskarnē — lietojumloga daļa, kas tiek izdalīta atsevišķam dokumentam. Tādās sistēmās kā *Microsoft Word* un *Word Perfect 6.0* dokumenta logs ir logs, kurā lietojumprogramma izveido vai modificē attiecīgo dokumentu.

**461 documentation**
**dokumentācija**
**документация**

Instrukcijas, specifikācijas, darbības apraksti, rokasgrāmatas un citi dokumenti, kas parasti ietilpst programmatūras vai aparatūras komplektā. Dokumentācija var būt noformēta tekstuālā vai elektroniskā formā.

**462 domain**
**domēns**
**домен**

1. Apgabals (ģeogrāfisks vai funkcionāls), kuram pieder tīklā *Internet* ieslēgtais dators. Parasti domēns norāda valsti, kurā atrodas dators (piem., Latvija (.lv), Kanāda (.ca) utt.). Dažkārt (īpaši ASV) domēns saistīts ar organizācijas raksturu, kurai pieder attiecīgais dators (piem., komerciāla (.com), izglītības (.edu), militāra (.mil) utt.).
2. Datu bāzu izstrādes un pārvaldības jomā ar šo terminu apzīmē kāda konkrēta lauka (atribūta) nozīmīgo vērtību kopu.

**463 domain name**
**domēna vārds**
**имя домена**

Vārdu sistēma, ko izmanto, lai identificētu konkrētu datoru tīklā *Internet*. Atšķirībā no *IP* adreses domēna vārds ir vieglāk iegaumējams nekā skaitliskā adrese. Sk. arī *Domain Name Service*.

**464 Domain Name Service**
**domēna vārdu servisprogramma**
**служба именования доменов**

Tīkla *Internet* programma, kas veic automātisku domēna vārdu translēšanu *IP* adresēs.

**465 DOS**

Sk. *disk operating system*.

**466 DOS extender**
***DOS* paplašinātājs**
**расширитель *DOS***

Programmatūra, kas, sākot ar personālajiem datoriem, kuri izveidoti uz mikroprocesoru *Intel 80286* bāzes, ļauj lietojumprogrammām izmantot atmiņu, kas pārsniedz 1 megabaitu. Lai gūtu pieeju paplašinātajai atmiņai, lietojumprogrammām jāstrādā aizsargrežīmā.

**467 DOS prompt**
***DOS* uzvedne**
**подсказка *DOS***

Operētājsistēmas *DOS* ziņojums, ko izspīdina displeja ekrānā un kas norāda, ka operētājsistēma ir gatava saņemt jaunu komandu.

**468 dot matrix**
**punktmatrica**
**точечная матрица**

Taisnstūra režģis vai matrica, ko izmanto, lai punktus izspīdinātu displeja ekrānā vai izdrukātu atbilstoši šablonam, ar kuru palīdzību veido rakstzīmes vai grafiskus attēlus.

**469 dot matrix printer**
**punktmatricas printeris**
**матричное печатающее устройство**

Sitienprinteris, kas ar punktmatricas palīdzību veido tekstu vai grafiskus attēlus, izdarot ar adatām sitienus pa kopējamo lenti. Lai paaugstinātu drukas kvalitāti, parasto 9 adatu vietā bieži izmanto 24 adatu matricprinterus, kas nodrošina augstāku drukātā attēla izšķirtspēju.

**470 dot pitch**
**punktiestatne**
**шаг точки**

Attālums starp vienas krāsas punktiem monitora ekrānā. Punktiestatne ir būtisks monitora raksturojums, kas nosaka tā maksimālo izšķirtspēju. Parasti šis parametrs ir 0,41 mm, 0,31 mm vai 0,28 mm. Jo mazāks tā lielums, jo lielāka ir monitora izšķirtspēja. Augstas izšķirtspējas monitoriem punktiestatne nepārsniedz 0,31 mm.

**471 double density disk**
**dubultblīvuma diskete, *DD* diskete**
**дискета двойной плотности**

Diskete, kurā var ierakstīt divreiz vairāk informācijas nekā agrāk lietotajā vienblīvuma disketē. Lai palielinātu informācijas apjomu, ierakstīšanai izmanto modificēto frekvences modulācijas tehnoloģiju. Parasti šī tipa 3,5" disketes ietilpība ir 720 kilobaiti.

**472 double strike**
**dubultsite**
**двойная печать**

Rakstzīmju divkārša drukāšana, lai padarītu to attēlus tumšākus.

**473 double-click**
**dubultklikšķis**
**двойной щелчок**

Divkārša peles pogas nospiešana un atlaišana pēc kārtas, nepārvietojot peles rādītāju (peli). Ar dubultklikšķi parasti veic tādas darbības kā logu atvēršanu un programmu palaišanu. Sk. arī *click*, *drag select*.

**474 double-sided disk**
**divpusēja diskete**
**двухсторонняя дискета**

Diskete vai cietais disks, kurā datus var ierakstīt abas pusēs.

**475 downloadable font**
**ielādējams fonts**
**загружаемый шрифт**

Noteikta burtveidola fonts, kas glabājas diskatmiņā un ko pārsūta uz printera atmiņu tad, kad tas nepieciešams kāda dokumenta drukāšanai.

**476 DPMS**

Sk. *display power management system*.

**477 DQDB**

Sk. *Distributed Queue Dual Bus*.

**478 drag and drop**
**vilkt un nomest**
**перетаскивание**

Paņēmiens, kas, izmantojot operētājsistēmas *Microsoft Windows* un firmas *Macintosh* programmas, ļauj izpildīt operācijas ar objektiem, neievadot komandas no tastatūras. Vilkšanu un nomešanu veic, ar peles palīdzību, pārvietojot objektu līdz tā sakrišanai ar tās lietojumprogrammas ikonu, kura izpilda nepieciešamo operāciju.

**479 drag select**
**velkatlase**
**выбор перетаскиванием**

Process, ko realizē, nospiežot peles pogu, turot to nospiestu un pārvietojot peles rādītāju pa displeja ekrānu. Velkatlasi pārtrauc, atlaižot peles pogu. Velkatlase dod iespēju atlasīt visus uzrādītos objektus starp pogas nospiešanas un atlaišanas punktiem.

**480 drag(ing)**
**vilkt; vilkšana**
**тащить; перетаскивание**

Attēla segmenta vai cita izvēlēta objekta pārvietošana pa displeja ekrānu, izmantojot peli. Vilkšanu izdara, norādot izvēlēto objektu, nospiežot peles pogu un pārvietojot peli vajadzīgajā virzienā. Ar peles pogas atlaišanu fiksē jauno objekta novietojumu.

**481 DRAM**

Sk. *dynamic random-access memory*.

**482 draw program**
**zīmēšanas programma**
**программа черчения**

Datorgrafikas programma attēlu zīmēšanai displeja ekrānā. Zīmēšanas programma veido attēlu, izmantojot tādus zīmējuma elementus kā līnijas, lokus un aploces, ko uzglabā atmiņā matemātisku formulu veidā. Zīmēšanas programmas izmanto vektorgrafiku, kas ļauj viegli atveidot attēlus dažādos lielumos.

**483 drawing tools**
**zīmēšanas rīki**
**инсрументарий рисования**

Krāsošanas programmā ietverts funkciju kopums, kas lietotājam dod iespēju zīmēt attēlus displeja ekrānā. Parasti zīmēšanas rīki redzami kā loku, līniju vai patvaļīgu figūru ikonas rīkjoslā.

**484 drive bay**
**diskdziņa niša**
**ниша дисковода**

Personālā datora korpusā izveidota atvere vai rezervēta telpa diskdziņu vai lenšdziņu instalēšanai.

**485 driver**
**dzinis, draiveris**
**драйвер**

Programma vai speciāla ierīce, kas vada dažādu procesu norisi vai ierīču darbību (piem., signālu pārsūtīšanu pa sakaru līnijām, diskdziņu, printeru u.c. ierīču funkcionēšanu), lai saskaņotu šo procesu izpildi vai ierīču sadarbību ar datoru. Sk. arī *device driver, line driver*.

**486 drop-down combination box**
**nolaižamais kombinācijlodziņš**
**раскрывающееся**
**комбинированное окно**

Kombinācijlodziņa paveids, kurā sarakstlodziņš ir apslēpts līdz brīdim, kad lietotājs veic noteiktu darbību, lai to padarītu redzamu. Sk. arī *combination box, drop-down list*.

487 **drop-down list**
**nolaižamais saraksts**
**раскрывающийся список**
Atlases lauks, kurā ir redzama tikai kārtējā izvēle. Pārējās izvēles ir apslēptas līdz brīdim, kad lietotājs veic noteiktu darbību, lai izspīdinātu sarakstlodziņu, kas satur pārējās izvēles. Sk. arī *drop-down combination box*.

488 **drop-down list box**
***nolaižamais sarakstlodziņš***
**раскрывающееся окно списка**
Komandu opciju saraksts, kas displeja ekrānā parādās kā teksta lodziņš, kamēr lietotājs atrod komandu, kas izsauc opciju saraksta nolaišanu. Pēc šī saraksta nolaišanas lietotājs var izvēlēties vienu no tā elementiem. Nolaižamais sarakstlodziņš, ietaupot ievērojamu ekrāna platību, nodrošina lietotāju ar iespēju izmantot daudzas opcijas.

489 **drop-down menu**
***nolaižamā izvēlne***
**раскрывающееся меню**
Izvēlne, kuras vārds ir izvēlņu joslā un kura parādās displeja ekrānā, ja lietotājs to izvēlējies, izmantojot peli vai tastatūru. Lai iniciētu kādu šajā izvēlnē norādīto darbību, ar peles vai tastatūras palīdzību atlases kursoru novieto uz vajadzīgā izvēlnes elementa un nospiež peles pogu vai ievadīšanas taustiņu.

490 **dual boot**
**duālā sāknēšana**
**двойственная самозагрузка**
Sāknēšana, kas dod iespēju uzsākt datora darbību, izmantojot vienu no divām datora atmiņā ievadītām operētājsistēmām.

491 **dual *in-line* package (DIP)**
**divrindu korpuss, *DIP* korpuss**
**корпус с двухрядовым расположением выходов**
Integrēto shēmu standartkonstrukcija, kurā ievietotās mikroshēmas izvadi izveidoti kā kontaktkājiņas, kas izvietotas abās garākajās šī moduļa malās. Sk. arī *single in-line memory modul, single in-line package*.

492 **dumb terminal**
**trulais terminālis**
**неинтеллектуальный терминал**
Terminālis, kas nevar patstāvīgi veikt datu apstrādi un kas kalpo kā attāla datora ievadizvades ierīce. Sk. arī *smart terminal*.

493 **dump**
**izmete**
**разгрузка**
Visa atmiņas satura vai tā daļas pārrakstīšana citā datu vidē (parasti — no iekšējās atmiņas ārējā) vai arī tā izdruka, lai nodrošinātos pret informācijas zudumiem iespējamu bojājumu vai avāriju gadījumā. Izmeti izmanto arī programmu atkļūdošanas procesā.

494 **DV-I**
Sk. *Digital Video-Interactive*.

495 **Dynamic Data Exchange (DDE)**
**protokols "Dinamiskā datu apmaiņa", protokols *DDE***
**динамический обмен данными**
Lietojumprocesu mijiedarbības protokols, ko nodrošina tādas operētājsistēmas kā *Microsoft Windows*, *Macintosh System 7*, *OS/2*. Pateicoties protokolam *DDE*, divas vai vairākas vienlaicīgi dar-

bojošās programmas var pārmaiņus nosūtīt cita citai datus un komandas bez lietotāja iejaukšanās. Pēdējā laikā dinamiskās datu apmaiņas mehānisms tiek aizstāts ar daudz spēcīgāku objektu sasaistes un iegultes protokolu.

**496 dynamic object**
**dinamiskais objekts**
**динамический объект**
Dokuments vai dokumenta daļa, ko ievieto izstrādājamajā dokumentā, izmantojot objekta sasaistes un iegultes tehniku. Sk. arī *embedded object*, *linked object*.

**497 dynamic RAM**
Sk. *dynamic random-access memory*.

**498 dynamic random-access memory (DRAM)**
**dinamiskā brīvpieejas atmiņa**
**динамическое запоминающее устройство с произвольной выборкой**
Datora atmiņa, kurā informācijas bitu uzglabāšanai izmanto mikroskopiskus kondensatorus. Tā kā šie kondensatori pakāpeniski izlādējas, tie vairākus simtus reižu sekundē ir jāuzlādē, lai saglabātu tajos ierakstīto informāciju. Sk. arī *static random-access memory*.

**499 dynamic resource allocation**
**dinamiskā resursu iedalīšana**
**динамическое распределение ресурсов**
Datora resursu iedalīšanas paņēmiens, kas paredz, ka, atkarībā no konkrētām vajadzībām, programmai piešķirtos resursus var mainīt programmas izpildes gaitā.

# E

**500 e-mail**
Sk. *electronic mail*.

**501 EBCDIC**
Sk. *Extended Binary Coded Decimal Interchange Code*.

**502 ECF**
Sk. *Enhanced Connectivity Facilities*.

**503 editing**
**rediģēšana**
**редактирование**
Dokumentu vai datņu satura, to izvietojuma vai attēlošanas formas mainīšana.

**504 editor**
Sk. *text editor*.

**505 EEPROM**
Sk. *Electrically Erasable Programmable ROM*.

**506 efficiency**
**efektivitāte**
**эффективность**
Pakāpe, kādā sistēma vai tās komponents izpilda savas funkcijas ar minimālo resursu patēriņu.

**507 EGA**
Sk. *Enhanced Graphics Adapter*.

**508 EIDE**
Sk. *Enhanced IDE*.

**509 EISA**
Sk. *Electronics Industry Standards Association; Extended Industry Standard Architecture*.

**510 ELD**

Sk. *electroluminiscent display*.

**511 Electrically Erasable Programmable ROM (EEPROM)**
**elektriski pārprogrammējamā lasāmatmiņa**
**электрически перепрограммируемая постоянная память**

Lasāmatmiņa, ko ar elektrisku signālu palīdzību var programmēt un vairākas reizes pārprogrammēt gan datorā, gan ārpus tā.

**512 electroluminiscent display**
**elektroluminiscences displejs**
**электролюминисцентный дисплей**

Displejs ar plakanu ekrānu, ko parasti veido, izmantojot divas caurspīdīgas plates, starp kurām ievietots plāns gaismu emitējoša pildījuma slānis. Vienā no platēm iestrādāti horizontāli, bet otrā vertikāli izvietoti vadi, kas veido koordinātu režģi. Padodot attiecīgiem diviem vadiem spriegumu, tiek lokalizēts kāds no režģa punktiem, kura apkārtne emitē gaismu. Šādus displejus izmanto portatīvajos datoros. Sk. arī *plasma display*.

**513 electronic data interchange (EDI)**
**elektroniska datu apmaiņa**
**электронный обмен данными**

Sistēma pasūtījumu, maksājumu, pavadzīmju vai citas (parasti komerciālas) informācijas pārsūtīšanai pa datoru tīklu vai telefona līnijām, izmantojot elektronisko pastu.

**514 electronic mail**
**elektroniskais pasts, e-pasts**
**электронная почта**

Ziņojumu pārsūtīšanas sistēma datoru tīklos. Elektroniskais pasts vai e-pasts ir īpašs pasta pakalpojumu veids, kas nodrošina to, ka tīklam pievienoto datoru lietotāji var nosūtīt un saņemt ziņojumus.

**515 electronic mail address**
**elektroniskā pasta adrese**
**адрес электронной почты**

Rakstzīmju virkne, kas precīzi identificē kādas personas pastkastītes atrašanās vietu. Tīklā *Internet* šo adresi veido pastkastītes vārds, kam seko zīme @ un datora domēna vārds, piem., gfricnov @edzi.lza.lv., kur gfricnov ir pastkastītes vārds, bet edzi.lza.lv. apzīmē datora (servera) atrašanās vietu — organizāciju un valsti.

**516 electronic mall**
**elektroniskā pasāža**
**электронные торговые ряды**

Virtuāla iepirkšanās vieta, kur var pārlūkot un iegādāties dažāda rakstura un dažādu firmu produktus un pakalpojumus.

**517 Electronics Industry Standards Association (EISA)**
**Elektronikas rūpniecības standartu asociācija, asociācija *EISA***
**Ассоциация стандартов электронной промышленности, ассоциация *EISA***

Asociācija, ko izveidojušas deviņas personālo datoru ražotāju grupas, lai ieviestu 32 bitu kopnes arhitektūru, kura

spētu konkurēt ar firmas *IBM* izstrādāto kopnes arhitektūru *MCA*. Sk. arī *Extended Industry Standard Architecture*.

**518 electrostatic printer**
**elektrostatiskais printeris**
**электростатический принтер**
Printeris, kas izmanto elektriski pretēji uzlādētas krāsvielas (tonera) daļiņas, lai izveidotu zīmējumu (tekstu) uz papīra. Pie šī tipa printeriem pieder, piem., lāzerprinteris, šķidro kristālu printeris un gaismas diožu printeris.

**519 embedded command**
**iegultā komanda**
**встроенная команда**
Tekstu apstrādē — komanda, kas ievietota kādā dokumentu datnē un kalpo kā instrukcija, lai, piem., izveidotu noteiktu lappuses izkārtojumu, mainītu fontus, veidotu pasvītrojumus. Iegultās komandas ir atkarīgas no izmantojamās tekstu apstrādes programmatūras.

**520 embedded hyperlink**
**iegultā hipersaite**
**встроенная гиперсвязь**
Hipersaite, kas ir iekļauta kādā teksta rindiņā.

**521 embedded object**
**iegultais objekts**
**вложенный объект**
Dokuments vai dokumenta daļa, ko veido, izmantojot vienu lietojumprogrammu, un kas tiek pilnībā ievietots (iegults) izstrādājamajā dokumentā, kuru veido, izmantojot citu lietojumprogrammu. Šo procesu īsteno ar objektu sasaistes un iegultes tehnikas starpniecību. Sk. arī *linked object*.

**522 EMM**
Sk. *Expanded Memory Manager*.

**523 emphasis**
**izcēlums**
**акцент**
Pasvītrošanas, kursīva, treknraksta u.tml. drukas stila izmantošana kāda vārda, frāzes vai teksta daļas uztveres atvieglošanai.

**524 EMS**
Sk. *Expanded Memory Specification*.

**525 emulation**
**emulācija; emulēšana**
**эмуляция**
1. Speciālas programmatūras un aparatūras izmantošana, lai ar kādu datoru izpildītu programmas, kas izstrādātas citas sistēmas datoram, un, izmantojot tos pašus datus, iegūtu identiskus rezultātus.
2. Dažādu protokolu piemērošana kādam noteiktam terminālim, lai nodrošinātu tā pievienošanu dažāda veida datu pārraides tīkliem.

**526 emulator**
**emulators**
**эмулятор**
Programmatūras un aparatūras kopums, kas veic emulāciju. Emulatora funkcionēšana saistīta ar kodu, formātu un procedūru pārveidošanu, kas ļauj dotās sistēmas darbību imitēt ar citas sistēmas līdzekļiem.

**527 enabling**
**iespējošana**
**допущение**
Aparatūras vai programmatūras darbības

traucējumu novēršana un funkcionēšanas atjaunošana, lai tā spētu veikt lietotājam nepieciešamās darbības. Sk. arī *disabling*.

**528 encapsulation**
**iekapsulēšana**
**инкапсуляция**

Datu struktūru un atsevišķu funkciju apvienošana vienā objektā, lai novērstu nevēlamus blakusefektus sistēmas darbībā. Izmantojot iekapsulēšanu, lielas objektorientētas programmas kļūst labāk lasāmas, jo visi dati un ar tiem saistītie kodi ir atrodami vienā vietā.

**529 encryption**
**šifrēšana**
**шифрование**

Datu un ziņojumu apstrādes process, ko veic datu sagatavotājs vai ziņojuma nosūtītājs, lai datus vai ziņojuma saturu nodrošinātu pret nesankcionētu izmantošanu. Lai šādus datus vai ziņojuma saturu varētu izmantot, jāveic tā dekodēšana. Sk. arī *ciphertext, cryptography, decryption*.

**530 End key**
**beigvietas taustiņš**
**клавиша конца**

Taustiņš, kas apvienots ar taustiņu 1 un izvietots firmas *IBM* un ar to saderīgas tastatūras cipartastatūras daļā, kā arī paplašinātās tastatūras rediģēšanas taustiņu blokā starp tastatūras pamatdaļu un cipartastatūru. Beigvietas taustiņam dažādās programmās mēdz būt atšķirīgas funkcijas. Visbiežāk tas pārvieto kursoru uz rindiņas beigām, uz labo apakšējo ekrāna stūri vai arī uz dokumenta beigām.

**531 end user**
**galalietotājs**
**конечный пользователь**

Lietotājs, kas savu uzdevumu risināšanai izmanto datoru sistēmas lietojumprogrammas.

**532 endnote**
**beigu vēre**
**ссылка**

Atsauce vai piezīme, kas izvietota dokumenta vai kādas tā daļas beigās, nevis lappuses apakšējā daļā. Daudzas teksta apstrādes programmas dod iespēju lietotājam izvēlēties beigu vēres izvietojumu.

**533 Enhanced Connectivity Facilities (ECF)**
**uzlabotās savienojamības iespējas, programmatūra *ECF***
**усовершенствование возможностей соединения**

Firmas *IBM* izstrādātā programmatūra, kas ļauj personālajiem datoriem, kuri strādā operētājsistēmas *DOS* vidē, pieprasīt un nolasīt datus no lieldatora, kā arī izpildīt lieldatora komandas.

**534 Enhanced Graphics Adapter (EGA)**
**videostandarts *EGA*; videoadapteris *EGA*, adapteris *EGA***
**усовершенствованный графический адаптер**

1. Videostandarts *IBM PC* tipa datoriem. Teksta režīmā šī standarta izšķirtspēja 16 krāsām ir 640×350, bet 4 krāsām — 720×350. Grafikas režīmā 16 krāsām tā augstākā izšķirtspēja ir 640×350. Šīs standarta iespējas ir lielākas par videostandarta *CGA* iespējām, bet nesasniedz *VGA* rādītājus.

2. Videoadapteris, kas atbilst videostandartam *EGA* un, izmantojot piemērotu monitoru, nodrošina tā prasību izpildi.

**535 Enhanced IDE (EIDE)**
**saskarne *EIDE***
**интерфейс *EIDE***

Pilnveidots saskarnes *IDE* standarts. Tas aptuveni 2 reizes palielina pārsūtīšanas ātrumu starp ārējo iekārtu un saskarnes adapteri un ļauj apkalpot četras ārējās iekārtas, piem., palielinātas ietilpības cietos diskus, *CD-ROM* ierīci, ierīci kopēšanai magnētiskajā lentē u.c. Turpretī saskarne *IDE* nodrošina tikai divu cieto disku apkalpošanu.

**536 enhanced keyboard**
**paplašinātā tastatūra**
**усовершенствованная клавиатура**

Tastatūra, kurai ir 101 vai 102 taustiņi un kuru firma *IBM* izveidoja personālo datoru *PC/AT* izstrādāšanas un izmantošanas laikā. Šis tastatūras taustiņu izvietojums kļuva arī par personālā datora *PS/2* standarttastatūru un līdz ar to faktiski par standartu vairumam tastatūru, kas saderīgas ar firmas *IBM* tastatūrām. Tās galvenā atšķirība no firmas *IBM* agrākajām tastatūrām ir tā, ka šai tastatūrai ir 12 funkcionālie taustiņi, kas izvietoti tās augšējā daļā, īpaši vadīšanas un alternēšanas taustiņi, kā arī kursora kustības vadīšanas un rediģēšanas taustiņu bloki, kas izvietoti starp tastatūras pamatdaļu un ciparu papildtastatūru. Sk. arī *AT keyboard, XT keyboard.*

**537 Enhanced Small Device Inter face (ESDI)**
**uzlabotā mazo ierīču saskarne, saskarne *ESDI***
**интерфейс *ESDI***

Cieto disku saskarnes standarts. Izmantojot šo saskarnes standartu, iespējams pārsūtīt 10—15 megabaitus sekundē, kas 2—3 reizes pārsniedz datu pārsūtīšanas ātrumu, ko nodrošina saskarne *ST-506*.

**538 Enter key**
**ievadīšanas taustiņš**
**клавиша ввода**

Tastatūras taustiņš, ko parasti izmanto, lai norādītu, ka pabeigta kāda rakstzīmju virknes ievade. Operētājsistēmās ievadīšanas taustiņš norāda, ka jāizpilda komanda. Tekstu apstrādes programmās ievadīšanas taustiņš parasti norāda rindkopas beigas, bet datu bāzu programmās — ka ieraksta ievade ir pabeigta un jāpāriet uz jaunu ierakstu.

**539 entry**
**ieeja, ieiešana; ievadīšana**
**вхождение**

1. Tabulas, saraksta, rindas vai citas organizētas datu struktūras elements.
2. Atsevišķa elementa vai elementu kopas ievade no termināļa.

**540 entry field**
**ievadlauks**
**поле ввода**

Displeja ekrānā laukums, kurā lietotājs var ierakstīt informāciju. Šī laukuma robežas parasti ir norādītas. Sk. arī *selection field.*

**541 environment**
**apkārtne; vide**
**окружение; среда**

Aparatūra un programmatūra, ar ko sadarbojas funkcionējošais objekts, piem., lietojumprogrammu izpildes apkārtni veido datori, to operētājsistēmas, datu kopas. Par šo terminu bieži runā šaurākā

nozīmē, attiecinot to, piem., uz oprētāj-sistēmas *Microsoft Windows* vidi. Datu pārraidei izmantojamo aparatūru un programmatūru parasti sauc par komunikāciju vidi.

**542 EPROM**
Sk. *Erasable Programmable Read-Only Memory.*

**543 erasable optical disc**
**dzēšamais optiskais disks**
**стираемый оптический диск**
Optisks disks, kurā datus var pārvietot, izmainīt un nodzēst tāpat kā magnētiskajā diskā. Šo disku atmiņas ietilpība sasniedz 1 gigabaitu, bet to izmaksas ir visai augstas un ātrdarbīgums ievērojami zemāks nekā cietajiem diskiem. Praktiski šī tipa diskus izmanto, ja nepieciešams saglabāt ļoti lielu papildinformācijas apjomu, piem., tehnisko dokumentāciju.

**544 Erasable Programmable Read-Only Memory (EPROM)**
**pārprogrammējamā lasāmatmiņa**
**стираемое программируемое постоянное запоминающее устройство**
Pusvadītāja lasāmatmiņa, kuras saturu lietotājs var dzēst, apstarojot to ar ultravioleto gaismu un pēc tam ierakstīt tajā jaunus datus ar speciālas ierīces — progammatora — palīdzību.

**545 error**
**kļūda**
**ошибка**
Atšķirība starp aprēķināto, izmērīto vai eksperimentāli iegūto mainīgā vērtību un tā patieso vērtību, kas var būt uzdota vai noteikta teorētiski.

**546 error checking**
**kļūdu pārbaude**
**контроль ошибок**
Process, kura gaitā datoru sistēma vai vairāki ar sakaru līdzekļiem savienoti datori pārbauda, vai datos, ar kuriem apmainās datori vai datoru sistēmas komponenti, nav kļūdu.

**547 error handling**
**kļūdu apdare**
**обработка ошибок**
Programmas apstrādes gaitā radušos kļūdu apstrāde. Kļūdu apstrādes programmatūras iespējas var būt ļoti dažādas — no vienkāršas lietotāja informēšanas par radušos kļūdu līdz automātiskai kļūdas vai tās izraisīto seku novēršanai.

**548 error model**
**kļūdu modelis**
**модель ошибок**
1. Modelis, ko izmanto, lai novērtētu vai prognozētu sistēmas faktiskās uzvedības novirzi no paredzētās uzvedības, piem., sakaru kanāla modelis, kas ļauj novērtēt šajā kanālā sagaidāmo pārraides kļūdu skaitu.
2. Modelis, ko izmanto, lai novērtētu vai paredzētu paliekošo kļūdu skaitu, nepieciešamo testēšanas laiku un citus līdzīgus raksturojumus, kas tiek izmantoti, izvērtējot programmatūru.

**549 escape character**
**atsoļa rakstzīme**
**знак перехода**
Vadības rakstzīme, ko bieži izmanto atsevišķi vai arī apvienojumā ar citām rakstzīmēm. Atsoļa rakstzīme parasti norāda atsoļa sekvences sākumu. Atsoļa rakstzīmes *ASCII* decimālais kods ir 27 un to ievada ar atsoļa taustiņa palīdzību. Sk. arī *Escape key, escape sequence.*

**550 Escape key**
**atsoļa taustiņš**
**клавиша перехода**

Tastatūras taustiņš ar atšķirīgu nozīmi dažādās programmās. Tas bieži tiek izmantots, lai atgrieztos iepriekšējā līmeņa hierarhiskajās izvēļņu struktūrās, kā arī lai izietu no programmas. Dažās programmās atsoļa taustiņš atver izvēlni, maina darba režīmu vai pārtrauc darbības izpildi.

**551 escape sequence**
**atsoļa sekvence**
**последовательность перехода**

Rakstzīmju sekvence, kas sākas ar atsoļa rakstzīmi un kam seko viena vai vairākas rakstzīmes, kuru apvienojums liek izpildīt komandu kādai ierīcei vai programmai, piem., mainīt krāsas displeja ekrānā, veidot speciālas uzvednes, mainīt printera iestatījumu no viena burtveidola uz otru.

**552 ESDI**

Sk. *Enhanced Small Device Interface*.

**553 Ethernet**
**tīkls *Ethernet***
**сеть *Ethernet***

Populāra lokālo tīklu arhitektūra, ko 70. gadu vidū izstrādāja firma *Xerox*. Tīklos *Ethernet* izmanto maģistrāles tīklu topoloģiju ar datu pārraides standartātrumu 10 megabiti sekundē. Kopējai maģistrālei pievienotajos datoros realizē *CSMA/CD* metodi. Uz tīkla *Ethernet* bāzes izstrādāts lokālo tīklu standarts *IEEE 802.3* un *ISO* specifikācija 8802/3, kur detalizēti aprakstītas dažādas tīkla *Ethernet* realizācijas atkarībā no izmantojamās datu nesējvides, piem., tīkls *Ethernet* ar tievo kabeli, vīto pāri, optisko kabeli u.c. Sk. arī *IEEE 802 Standards*.

**554 EtherTalk**
**tīkls *EtherTalk***
**сеть *EtherTalk***

Firmas *Apple* personālo datoru lokālais tīkls, kas veidots pēc tīkla *Ethernet* principiem. Tāpat kā tīklā *Ethernet*, arī šeit datorus savieno ar koaksiālo kabeli, kas nodrošina datu pārraidi ar ātrumu 10 megabiti sekundē. Datorus pieslēdz tīklam, izmantojot speciālas adaptera kartes. Kā tīkla operētājsistēmu izmanto firmas *Apple* izstrādāto programmatūru *AppleShare*.

**555 Excel**
**programma *Excel***
**программа *Excel***

Firmas *Microsoft* izveidota plaša izklājlapu apstrādes programma, kas lietojama firmas *IBM* un firmas *Apple* datoriem. Tā var apvienot vairākas izklājlapas un nodrošina lietotājiem lielu grafisko līdzekļu klāstu.

**556 EXE file**
***EXE* datne**
***EXE* файл**

*EXE* datne ir operētājsistēmas *DOS* vai *OS/2* datne, kuras vārda paplašinājumā ietilpst sufikss "*EXE*". Šo sufiksu izmanto, lai apzīmētu datni, kas satur pārvietojamu izpildāmu programmu.

**557 executable program**
**izpildāma programma**
**выполняемая программа**

Datora programma, kas gatava palaišanai noteiktā datorā. Lai programma būtu izpildāma, tā iepriekš jāpārtranslē attiecīgā datora mašīnvalodā.

**558 exit**
**izeja; iziešana**
**выход**

Programmas vai cikla izpildes pabeigšana. Parasti programmai ir viens iziešanas punkts, kurā tiek pabeigta programmas izpilde. Dažkārt programmai var būt vairāki iziešanas punkti, kas atbilstoši dažādiem nosacījumiem ļauj pārtraukt tās izpildi.

**559 expandability**
**izvēršamība**
**расширяемость**

Datora vai datoru sistēmas funkcionālo iespēju paplašināšana, pievienojot tai dažādus moduļus vai ierīces.

**560 expanded memory**
**izvērstā atmiņa**
**наращиваемая память**

*IBM PC* tipa vai ar tiem saderīgajos personālajos datoros pamatatmiņas paplašinājums, kura izmantošanu panāk, ar speciālas aparatūras un programmu starpniecību apvienojot atsevišķas atmiņas shēmas vienotā adrešu blokā.

**561 Expanded Memory Manager (EMM)**
**izvērstās atmiņas pārvaldnieks, draiveris *EMM***
**диспетчер отображаемой памяти**

Programmatūra, kas nodrošina izvērstās atmiņas izmantošanu. Personālajos datoros, kas izveidoti uz mikroprocesoru *Intel 8086* un *80286* bāzes, šī programmatūra jālieto kopā ar izvērses plati, bet datoros, kas izveidoti uz mikroprocesoru *Intel 80386* un *80486* bāzes, papildaparatūra nav nepieciešama.

**562 Expanded Memory Specification (EMS)**
**izvērstās atmiņas specifikācija, specifikācija *EMS***
**спецификация отображаемой памяти**

Atmiņas izvēršanas paņēmienu un līdzekļu specifikācija, kas operētājsistēmā *DOS* strādājošiem personālajiem datoriem kopumā nodrošina iespēju izmantot izvērsto atmiņu programmu izpildei. Šo specifikāciju dažkārt sauc arī par firmu *Lotus, Intel, Microsoft (LIM)* izvērstās atmiņas specifikāciju, jo tā izstrādāta minēto trīs firmu sadarbības rezultātā. Sk. arī *expanded memory*.

**563 expansion board**
**izvērses plate, paplašināšanas plate**
**расширительная плата**

Drukātā plate, ko pievieno datora kopnei, lai paplašinātu datora funkcionālās iespējas un palielinātu veiktspēju.

**564 expansion bus**
**izvērses kopne, paplašināšanas kopne**
**расширительная шина**

Īpašas datora kopnes un protokolu kopums, kas ļauj paplašināt datora funkcijas un resursus, pievienojot tam izvērses plates un perifērijas ierīces.

**565 expansion card**

Sk. *expansion board*.

**566 expansion slot**
**izvērses slots, paplašināšanas slots**
**гнездо для расширительных модулей, расширительное гнездо**

Kontaktligzda, ar kuras starpniecību datoram var pievienot izvērses plates, lai papildinātu tā funkcionālās iespējas. Daudziem personālajiem datoriem ir vairāki izvērses sloti, kas paredzēti atmiņas palielināšanai, grafikas apstrādes iespēju paplašināšanai vai kādu speciālu ierīču pievienošanai.

567 **export**
**eksports**
**экспорт**

Datnes pārvēršana no formāta, kādā to izmanto noteikta programma vai datu apstrādes sistēma, citā formātā, kas nepieciešams, lai to varētu izmantot cita programma vai sistēma. Vairums programmu var viegli eksportēt *ASCII* formāta datnes, jo gandrīz visas programmas tās var lasīt un izmantot.

568 **extendability**
**paplašināmība**
**расширяемость**

Sk. *expandability*.

569 **Extended Binary Coded Decimal Interchange Code (EBCDIC)**
**informācijas apmaiņas paplašinātais binārī decimālais kods, kods *EBCDIC*, *EBCDIC***
**расширенный двоично-десятичный код обмена информацией, код *EBCDIC***

Standartizēts astoņu bitu kods, ko plaši izmanto datortehnikā (piem., firmas *IBM* lieldatoros un minidatoros). Ar šo kodu var identificēt 256 dažādas rakstzīmes un vadības simbolus. Atšķirībā no koda *ASCII*, koda *EBCDIC* tabulā alfabēta burti nav izvietoti nepārtrauktā secībā. Nav arī vienozīmīgas atbilstības starp kodos *EBCDIC* un *ASCII* izmantotajām kodu kombinācijām.

570 **eXtended Graphics Array (XGA) videostandarts *XGA*; videoadapteris *XGA*, adapteris *XGA***
**адаптер расширенной графики**

1. Firmas *IBM* videostandarts, kas izstrādāts, lai aizstātu novecojušo videostandartu *8514/A*. Teksta režīmā šī standarta izšķirtspēja 16 krāsām ir 1056×768, bet grafikas režīmā 65536 krāsām — 640×480 un 256 krāsām — 1024×768. Videostandarts *XGA* ir saderīgs ar videostandartu *VGA*.
2. Videoadapteris, kas atbilst videostandartam *XGA* un kas, izmantojot pietiekamu papildatmiņu un piemērotu monitoru, nodrošina tā prasību izpildi.

571 **Extended Industry Standard Architecture (EISA) paplašinātā industriālā standartarhitektūra, arhitektūra *EISA***
**архитектура *EISA***

Personālo datoru kopnes arhitektūras standarts, ko, konkurējot ar arhitektūru *MCA*, 1988. g. ieviesa Elektronikas rūpniecības standartu asociācija. Arhitektūra *EISA* dod iespēju paplašināt arhitektūras *ISA* datu kopni līdz 32 bitiem. Datu pārraide pa šādas arhitektūras kopni ir sinhrona un pa to var sadarboties vairāki kopnes vedēji. Arhitektūras *EISA* slotos var ievietot arī arhitektūrai *ISA* atbilstošas kartes. Kopnes, kas izveidotas atbilstoši šai arhitektūrai, bieži sauc par *EISA* kopnēm.

**572 extended memory**
**paplašinātā atmiņa**
**расширенная память**

*PC/AT* un *IBM PS/2* tipa datoros, kas izmanto *Intel 80286* un jaunākus mikroprocesorus, pamatatmiņas paplašinājums, kuru var izmantot operētājsistēmas *MS-DOS* vidē tikai ar speciālas programmatūras starpniecību, bet operētājsistēmu *OS/2* un *UNIX* vidē — bez papildprogrammu palīdzības.

**573 eXtended Memory Specification (XMS)**
**paplašinātās atmiņas specifikācija, specifikācija *XMS***
**спецификация расширенной памяти**

Atmiņas paplašināšanas paņēmienu un līdzekļu specifikācija, kas operētājsistēmas *DOS* vidē strādājošajiem personālajiem datoriem kopumā nodrošina iespēju izmantot paplašināto atmiņu. Sk. arī *extended memory, HIMEM.SYS.*

**574 eXtended Technology (XT)**
**tehnoloģija *XT***
**технология *XT***

Tehnoloģija, kas izmantota, izstrādājot pirmo firmas *IBM* personālo datoru ar cieto disku. Centrālā procesora funkcijas veica mikroprocesors *Intel* 8086 vai *8088*. Šīs klases datoru atmiņas apjoms nepārsniedza 1 megabaitu, kas radīja ievērojamas neērtības to lietotājiem. Sk. arī *Advanced Technology*.

**575 extension**
**paplašinājums**
**расширение**

Sk. *file name extension.*

**576 external command**
**ārējā komanda**
**внешняя команда**

Operētājsistēmu *DOS* un *OS/2* programma, ko ielādē atmiņā un kas tiek izpildīta tikai tad, kad tās vārdu ievada ar sistēmas uzvednes palīdzību.

**577 external data bus**
**ārējā datu kopne**
**внешняя шина данных**

Sakaru kanālu kopums, kas nodrošina datu apmaiņu starp centrālo procesoru un citiem mātesplates komponentiem, ieskaitot brīvpieejas atmiņu.


**578 facsimile**
**faksimils**
**факсимиле**

Sk. *fax.*

**579 failure**
**kļūme, atteice**
**сбой, отказ**

Situācija, kad datu apstrādes sistēma vai kāda tās sastāvdaļa bojājumu dēļ daļēji vai pilnīgi zaudē spēju izpildīt tai paredzētās funkcijas.

**580 FAQ**

Sk. *Frequently Asked Questions.*

**581 fast Ethernet**
**ātrais *Ethernet* tīkls**
**быстрая сеть *Ethernet***

Lokālā datoru tīkla *Ethernet* arhitektūras versija, kas nodrošina datu pārraides ātrumu līdz 100 megabitiem sekundē,

saglabājot standartā *IEEE 802.3* paredzēto paketes formātu un izmantojot vides pieejas metodi *CSMA/CD*. Kā datu nesējvidi ātrajā *Ethernet* tīklā izmanto vīto pāri vai optisko kabeli. Sk. arī *IEEE 802 Standards*.

582 **FAT**
Sk. *file allocation table*.

583 **fatal error**
**fatāla kļūda**
**фатальная ошибка**
Kļūda, kuras dēļ sistēma vai tās komponents kļūst pilnīgi nespējīgs veikt savas funkcijas. Fatāla kļūda programmā parasti rada tās priekšlaicīgu pārtrauci vai avāriju, kas saistīta ar datu neatgriezenisku zaudēšanu.

584 **FatBits**
**treknbiti**
**укрупненные биты**
Ekrānā stipri palielināti pikseļi. Sākotnēji palielināšanu veica programma *MacPaint*, nodrošinot mazu zīmējuma daļu palielināšanu un modificēšanu secīgi pa vienam pikselim. Šo mērogmaiņas iespēju tagad nodrošina vairums krāsošanas programmu.

585 **fault**
**bojājums, defekts**
**неисправность, ошибка**
Stāvoklis, kas rada kādas sistēmas vai tās sastāvdaļas funkciju nepareizu izpildi.

586 **fault tolerance**
**bojājumpiecietība**
**отказоустойчивость**
Aparatūras vai programmatūras spēja pareizi funkcionēt gadījumos, kad sistēmas iekārtās ir kļūme vai bojājums.

587 **fax**
**fakss**
**факс, факсимильная связь**
1. Sistēma dokumentu (tekstu, zīmējumu, shēmu, attēlu) pārraidei pa sakaru līnijām. Raidierīce nosūtāmo dokumentu skenē un pārvērš analogsignālā vai ciparsignālā, bet uztvērējierīce no šī signāla atveido dokumenta kopiju. Sk. arī *scanner*.
2. Teksta vai grafisku attēlu pārraide.
3. Teksta vai grafisku attēlu kopija.

588 **fax machine**
**faksa mašīna**
**факсмашина**
Sk. *fax*.

589 **fax modem**
**faksmodems**
**факсмодем**
Specializēts modems, kas dod iespēju izmantot personālo datoru faksu pārraidīšanai un saņemšanai.

590 **FCB**
Sk. *file control blok*.

591 **FDDI**
Sk. *Fiber Distributed Data Interface*.

592 **feathering**
**atstarpošana**
**растяжка строки**
Vienādu atstarpju izveidošana starp rindiņām lappusē vai kolonnā, lai realizētu vertikālo taisnošanu.

593 **female connector**
**sievišķais savienotājs**
**розетка разъема**
Datora kabeļa pieslēgvietas konstrukcija, kuras ligzdas izvietotas atbilstoši vīrišķā savienotāja spraudņu izvietojumam.

**594 FF**
Sk. *form feed.*

**595 Fiber Distributed Data Interface (FDDI)**
**optiskās šķiedras dalīto datu saskarne, standarts *FDDI*, *FDDI***
**распределенный интерфейс оптоволоконной передачи**
Optisko šķiedru datu pārraides tīklu standartu kopums, ko izstrādājis Amerikas Nacionālais standartu institūts (*ANSI*). *FDDI* standartā tālāk attīstītas marķiergredzena tīkla pamatidejas. Standarts *FDDI* nosaka saskarnes un protokolus ātrdarbīgiem pilsēttīkliem, kas nodrošina augstas izšķirtspējas grafikas un diskrētās videoinformācijas ātru pārsūtīšanu. Sk. arī *Copper Distributed Data Interface.*

**596 fiber optic cable**
**optiskais kabelis**
**волоконно-оптический кабель**
Kabelis, kas izveidots no optiskajām šķiedrām. Optiskajos kabeļos informācijas pārsūtīšanai izmanto gaismas impulsus. Šī tipa kabeļi ļauj pārsūtīt lielu datu daudzumu un ir nejūtīgi pret elektriskajiem traucējumiem. Tos parasti izmanto datu pārraides pamattīkla veidošanai.

**597 FidoNet**
**tīkls *FidoNet*, *FidoNet***
**сеть *FidoNet***
*FidoNet* ir nekomercionāls amatieru datoru tīkls, kura mērķis ir nodrošināt lietotājiem gan pieeju elektroniskajam pastam, gan dalību elektroniskajās konferencēs, gan arī privāto saraksti. No šī tīkla, izmantojot vārtejas, ir iespējams nosūtīt elektroniskā pasta ziņojumus jebkuram citam tīklam, t.sk. tīklam *Internet*. Pēdējos gados vērojama šī tīkla strauja attīstība, kas izskaidrojama ar relatīvi vienkāršu pieeju tīklam *FidoNet*. Lai sāktu darbu šajā tīklā, ir nepieciešams tikai dators, modems un telefona līnija. Kopš 1993. g. vērojama intensīva šī tīkla darbība arī Latvijā.

**598 field**
**lauks**
**поле**
Ieraksta daļa ar patstāvīgu funkcionālu nozīmi, ko datora programma apstrādā kā atsevišķu datu elementu (piem., komandas adreses lauks, operanda lauks, ziņojuma datu lauks u.c.).

**599 field definition**
**lauka definīcija**
**определение поля**
Noteikumi, ko izmanto datu bāzes pārvaldības programma un kas norāda, kāda veida informāciju var ievadīt datu laukā. Sk. arī *data type*, *field template.*

**600 field name**
**lauka vārds**
**имя поля**
Unikāls vārds, kas palīdz lietotājam noteikt lauka saturu un ko datu bāzes pārvaldības programma izmanto, lai identificētu noteiktu datu lauku.

**601 field template**
**lauka veidne**
**трафарет поля**
Definīcija, kas nosaka datu veidu, ko var ierakstīt datu laukā. Mēģinot ierakstīt lauka veidnei neatbilstošus datus, datu bāzes pārvaldības programma inicie kļūdas ziņojuma parādīšanos displeja

ekrānā. Lauka veidnes neļauj lietotājam ierakstīt datu bāzē tai nepiemērotu informāciju.

602 **field type**
**lauka tips**
**тип поля**
Datu lauka paveids, ko nosaka tajā ierakstāmo datu tips.

603 **FIF**
Sk. *fractal image format.*

604 **file**
**datne, fails**
**файл**
Datu kopa, tekstuāls vai grafisks dokuments, ko glabāšanas, pārsūtīšanas vai apstrādes procesā uzskata un identificē kā vienotu veselumu un kas parasti sastāv no vienādas struktūras ierakstiem.

605 **file allocation table (FAT)**
**datņu iedales tabula, tabula *FAT***
**таблица распределения файлов**
Operētājsistēmu *DOS* un *OS/2* datņu sistēmas sastāvdaļa, kurā tiek glabāti visu diskā ierakstīto datņu lielumi un izvietojums, jo datne var būt ierakstīta vairākos savstarpēji nesaistītos atmiņas apgabalos.

606 **file attribute**
**datnes atribūts**
**атрибут файла**
Datnei pievienota iezīme, kas apraksta un nosaka tās statusu vai izmantošanas veidu. Datnes atribūts, piem., var norādīt tās piederību lasāmajām vai arhīva datnēm, kā arī to, vai tā ir sistēmas, direktorija vai apslēptā datne.

607 **file backup**
Sk. *backup.*

608 **file compression**
**datnes saspiešana**
**сжатие файла**
Datnes uzglabāšanai nepieciešamās atmiņas telpas samazināšana.

609 **file control block (FCB)**
**datnes vadības bloks**
**блок управления файлами**
Mazs atmiņas bloks, ko operētājsistēma īslaicīgi piešķir, lai tajā varētu ierakstīt informāciju par lietojumam atvērto datni. Datnes vadības blokā parasti ir datnes identifikators, tās izvietojums uz diska, kā arī norāde par pēdējo lietotāja izmantoto pozīciju datnē.

610 **file conversion**
**datnes konversija**
**преобразование файла**
Datnes datu pārvēršana no viena formāta otrā, nemainot tās saturu.

611 **file deletion**
**datnes dzēšana**
**удаление файлов**
Process, kurā datne tiek padarīta par nelietojamu. Izšķir divus datņu dzēšanas veidus: fizisko un loģisko. Loģiskā dzēšana padara datni lietotājam nemanāmu, bet saglabā tās atkopšanas iespējas. Fiziskā dzēšana saistīta ar datnes magnētiskā ieraksta likvidēšanu.

612 **file extension**
**datnes paplašinājums**
**расширение файла**
Sk. *file name extension.*

613 **file format**
**datnes formāts**
**формат файла**
1. Jebkurš šablons vai standarts, ko izmanto datu uzkrāšanai diskā, attēlošanai displeja ekrānā vai drukāšanai.
2. Kodēta informācija par datnes uzbūvi.

614 **file fragmentation**
**datņu fragmentēšana**
**фрагментация файлов**
Datņu izvietošana dažādos diskešu vai cieto disku sektoros, kas neatrodas viens otram blakus. Datņu fragmentēšanu parasti rada vairākkārtēja to dzēšana un ierakstīšana un tā būtiski samazina diska darbības efektivitāti, jo pagarina lasīšanas/rakstīšanas galviņas parvietošanās laiku, kas nepieciešams vajadzīgās datnes atrašanai. Sk. arī *defragmentation*.

615 **file handle**
**datņturis**
**дескриптор файла**
Operētājsistēmās *MS-DOS*, *OS/2* un *Windows NT* lietots skaitlis, kurā sistēmas pārvērtušas datnes vārdu, ko tās izmanto, lai realizētu pieeju datnei.

616 **file header**
**datnes galvene**
**заголовок файла**
Sk. *header*.

617 **file layout**
**datnes izkārtojums**
**описание структуры файла**
Datnes ierakstu organizācijas veids. Datnes izkārtojumā var ietilpt arī ierakstu struktūras apraksti.

618 **file locking**
**datnes slēgšana**
**блокировка файла**
Laiksakritības vadības metode, kas nodrošina datu integritāti. Datnes slēgšana novērš vairāku lietotāju vienlaicīgu pieeju datnei un tās modificēšanu.

619 **file maintenance**
**datnes uzturēšana**
**сопровождение файла**
1. Datnes atjaunināšana, pievienojot, izmainot vai izslēdzot no tās datus.
2. Diskatmiņas periodiska sakārtošana. Dati, kurus darba procesā nepieciešams mainīt, parasti fiziski nonāk dažādās diskatmiņas vietās, un tie ar speciālu programmu palīdzību jāpārgrupē, lai iegūtu nepārtrauktu datni.

620 **file management program**
**datņu pārvaldības programma**
**программа управления файлами**
Programma darbam ar datnēm, direktorijiem un diskiem. Pārskatot esošās datnes, šī programma nodrošina iespēju iepazīties ar disku direktoriju saturu. Izmantojot programmas izvēlņu komandas, var pārvietot un kopēt datnes, kā arī veikt citus uzdevumus, kas saistīti ar disku izmantošanas uzlabošanu un lietotāju datu aizsargāšanu.

621 **file manager**
**datņu pārvaldnieks**
**администратор файлов**
Programmatūra, kas veic datņu pārvaldību, nodrošinot datņu dzēšanu, kopēšanu, pārvietošanu, pārdēvēšanu un pārlūkošanu, kā arī direktoriju izveidošanu un pārvaldību.

622 **File Manager**
**lietojumprogramma *File Manager*, *File Manager***
**администратор файлов**

Datņu, direktoriju un disku pārvaldībai paredzēta lietojumprogramma *Microsoft Windows* vidē. Lietojumprogramma *File Manager* displeja ekrānā parāda diska direktoriju struktūru un tajos esošās datnes.

### 623 file name
### datnes vārds
### имя файла

Rakstzīmju secība, ko izmanto konkrētas datnes identificēšanai. Ar datnes vārda palīdzību lietotājs gan saglabā, gan pieprasa informāciju.

### 624 file name extension
### datnes vārda paplašinājums
### расширение имени файла

Sufikss (parasti trīs burti), ko pievieno operētājsistēmu *DOS* un *OS/2* datņu nosaukumiem un kas norāda datnes tipu, piem., izpildāmajām datnēm sufikss var būt *EXE*, *COM* vai *BAT*. Sk. arī *file allocation table*.

### 625 file recovery
### datnes atkopšana
### восстановление файла

Nodzēstu vai bojātu disku datņu atjaunošana. Datņu atkopšanai tiek izmantotas specializētas programmas.

### 626 file server
### datņu serveris
### файловый сервер

Specializēts servera veids, kas nodrošina pieeju koplietošanas datnēm. Parasti tas ir viens no tīkla datoriem, kurā tiek uzkrātas datnes, lai tās varētu izmantot klientu datori. Lielākos tīklos datņu serveris darbojas speciālas operētājsistēmas vadībā, bet mazākos tas var strādāt ar parasto personālā datora operētājsistēmu, kas papildināta ar vienādranga tīklu operētājsistēmām, piem., ar operētājsistēmu *Personal NetWare* un operētājsistēmu *LANtastic*.

### 627 file sharing
### datnes koplietošana
### совместное использование файла

Situācija datoru tīklā, kurā divu vai vairāku uzdevumu risināšanai vienlaicīgi izmanto vienu un to pašu datni, kas parasti tiek uzglabāta vai nu centrālajā datorā vai tīkla datņu serverī.

### 628 file size
### datnes lielums
### размер файла

Datnes garums baitos.

### 629 file structure
### datnes struktūra
### файловая структура

Datnes vai kopīgi apstrādājamas datņu kopas apraksts, kurā ietilpst datņu izkārtojums un katras apskatāmās datnes izvietojums.

### 630 file system
### datņu sistēma
### система файлов

1. Operētājsistēmas struktūra, kas nosaka, kā datnes tiek veidotas, nosauktas un saglabātas. Datņu sistēma sastāv no datnēm, direktorijiem un informācijas, kas nepieciešama, lai atrastu šos elementus un realizētu tiem pieeju.
2. Operētājsistēmas sastāvdaļa, kas translē lietojumprogrammu pieprasījumus darbībām ar datnēm.

631 **File Transfer Protocol (FTP) datņu pārsūtīšanas protokols, protokols *FTP* протокол передачи файлов**
Protokolu *TCP/IP* sistēmas sastāvdaļa, kas aptuveni atbilst atvērto sistēmu sadarbības bāzes etalonmodeļa lietojumslānim. Izmantojot protokola *TCP* pakalpojumus, protokols *FTP* ļauj tīkla lietotājiem apskatīt attālu datoru direktorijus, nolasīt, pārsūtīt vai atjaunot to datnes.

632 **file viewer datņu skatītājs программа просмотра файла**
Programmatūra, kas datnes saturu uz displeja ekrāna attēlo tādā pašā veidā, kādā to būtu attēlojis tas lietojumprocess, kurš to izveidojis.

633 **film recorder filmu ierakstītājs слайд-принтер**
Personālā datora izvadierīce, kas pārkopē attēlus no grafikas datnes uz fotofilmu.

634 **filter filtrs фильтр**
1. Programma, piem., kārtošanas palīgprogramma, kas maina elementu kārtību datu sekvencēs, vai pārvēršanas palīgprogramma, kas datus, tekstu vai grafiku pārvērš no viena formāta citā.
2. Šablons vai maska noteiktu datu atlasei, piem., datu bāzu filtrs, kas var atsijāt datus, kuri nav saskaņoti ar iepriekš doto specifikāciju kopu.
3. Sakaru tehnikā — programmatūra vai aparatūra, kas laiž cauri zināmus signāla elementus, bet atsijā vai minimizē citus.

635 **finger pirksts палец**
Tīkla *Internet* programma, kas ļauj iegūt informāciju par lietotāju, kuram ir elektroniskā pasta adrese (lietotāja pilnu vārdu, nodarbošanos, adresi, pēdējā seansa laiku, termināļa adresi u.c.).

636 **fingerprint reader pirkstu nospiedumu lasītājs считыватель отпечатков пальцев**
Skeneris, kas nolasa pirkstu nospiedumus un ievada tos datorā salīdzināšanai ar datu bāzē glabājamajiem attēliem, tādējādi nodrošinot kādas personas identifikāciju. Pirkstu nospiedumu lasītājus lieto, piem., lai aizsargātu datnes no nesankcionētas izmantošanas.

637 **firewall ugunsmūris межсетевое устройство защиты**
Drošības sistēma, kas paredz speciāli programmēta datora ievietošanu starp kādas organizācijas lokālo datoru tīklu un tīklu *Internet*. Ugunsmūris aizsargā šo lokālo tīklu no nesankcionētas tīkla *Internet* lietotāju pieejas, bet apgrūtina aizsargātā tīkla lietotājiem tīkla *Internet* pakalpojumu izmantošanu.

638 **firmware programmaparatūra программно-аппаратное обеспечение**
Programmatūra vai dati, kas ierakstīti pastāvīgai lietošanai dažāda tipa lasāmatmiņās un veido nedalāmu programmatūras un aparatūras apvienojumu.

639 **fixed disk**
**fiksētais disks**
**фиксированный диск**
Cietais disks, ko nevar noņemt no piedziņas mehānisma un ko izmanto vairumā personālo datoru.

640 **fixed head disk**
**fiksētu galviņu disks**
**диск с фиксированными головками**
Cietais disks ar katram diska celiņam piekārtotām nekustīgām lasīšanas/rakstīšanas galviņām. Šāda konstrukcija ļauj ievērojami samazināt piekļuves laiku, jo novērš galviņu pārvietošanas izraisītos laika zudumus.

641 **fixed pitch**
**fiksētā iestatne**
**фиксированное размещение**
Vienāda platuma rakstzīmju aizņemto vietu izmantošana drukātā tekstā. Fiksētā iestatne aizņem lielāku laukumu, salīdzinot ar proporcionālo iestatni, bet ir ērti lietojama kolonnu drukāšanai.

642 **flash memory**
**zibatmiņa**
**флеш-память**
Elektriski pārprogrammējamai lasāmatmiņai līdzīga energoneatkarīga atmiņa, kas operē nevis ar atsevišķiem bitiem, bet ar veseliem datu blokiem. Raksturīgākā zibatmiņas īpatnība ir tā, ka visu tās saturu var dzēst vienlaicīgi.

643 **flat address space**
**izplātā adrešu telpa**
**двумерное адресное пространство**
Adrešu telpa, kurā katru atmiņas vietu nosaka atšķirīgs unikāls skaitlis. Atmiņas adrešu numerācija sākas ar 0 un secīgi pieaug par 1. Sk. arī *segmented address space*.

644 **flat file**
**izplātā datne**
**плоский файл**
Datne, kas nav fizikāli saistīta ar citām datnēm un nesatur norādes uz tām. Izplātajai datnei nepiemīt hierarhiska struktūra.

645 **flat file directory**
**izplātais datņu direktorijs**
**двумерный каталог файлов**
Direktorijs, kurā nav apakšdirektoriju, bet ir tikai datņu saraksts.

646 **flat panel display**
**plakanais displejs**
**плоский дисплей**
Plāns displeja ekrāns, ko lieto klēpjdatoros un piezīmjdatoros. Plakanā displeja izgatavošanai izmanto šķidro kristālu, gāzes plazmas, elektroluminiscences vai plāno plēvju tranzistoru tehnoloģiju. Sk. arī *liquid crystal display*, *plasma display*, *thin film transistor screen*.

647 **flat screen**
**plakanais ekrāns**
**плоский экран**
Sk. *flat panel display*.

648 **flat-file database management program**
**izplātās datnes datu bāzes pārvaldības programma**
**программа управления базой данных плоского файла**
Datu bāzes pārvaldības programma, kas glabā, organizē un izgūst informaciju

vienlaicīgi tikai no vienas datnes. Šādām programmām nepiemīt relāciju datu bāzu pārvaldības iespējas.

649 **flexibility**
**pielāgojamība**
**гибкость**
Aparatūras vai programmatūras spēja piemēroties mainīgiem apstākļiem vai izpildāmajiem uzdevumiem.

650 **flicker**
***mirgoņa***
**мерцание**
Vizuāls traucējums (piem., attēla spilgtuma maiņa), kas parādās tādu monitoru ekrānos, kam zems atsvaidzināšanas ātrums. Parasti šie traucējumi ir raksturīgi monitoriem ar rindpārlēces izvērsi.

651 **floating point processor**
**peldošā komata procesors**
**процессор с плавающей запятой**
Līdzprocesors,kas specializēts ātru operāciju ar peldošo komatu (punktu) izpildei. Peldošā komata procesors var būt izveidots kā atsevišķa mikroshēma, ko sistēmai var pievienot, izmantojot speciālu ligzdu, vai arī tas var būt apvienots vienā mikroshēmā ar mikroprocesoru (piem., *Intel 80486DX, Motorola 68040, Pentium*). Sk. arī *math coprocessor*.

652 **floppy disk**
**diskete, lokanais disks**
**гибкий диск, дискета**
Aizsargapvalkā ievietots plastmasas disks, kura viena vai abas virsmas pārklātas ar magnētisku materiālu datu ierakstīšanai. Disketes personālajos datoros izmanto kā lētu liela apjoma portatīvu atmiņu. Diskešu parastais diametrs ir 5,25 un 3,5 collas.

653 **flush left**
**kreisā sabīde**
***смещение влево***
Teksta horizontāla izlīdzināšana attiecībā pret kreiso dokumenta malu, panākot, ka katra rindiņa sākas vienā un tai pašā pozīcijā. Kreisajā sabīdē rindiņu garums var būt dažāds.

654 **flush right**
**labējā sabīde**
**смещение вправо**
Teksta horizontālā izlīdzināšana attiecībā pret labo dokumenta malu, panākot, ka katra rindiņa beidzas vienā un tai pašā pozīcijā, bet rindiņas var sākties dažādās pozīcijās. Sk. arī *ragged margin*.

655 **folder**
**mape**
**папка**
Kopā uzglabājamu datņu un programmu grupa, kuras apzīmēšanai izmanto vienu vārdu vai grafisku attēlu (ikonu). Mape ir jēdziens, kas ekvivalents direktorijam. Mapē var būt datnes, kā arī citas mapes, un to izmanto kā līdzekli programmu un datu izvietošanai diskatmiņā.

656 **font**
**fonts**
**шрифт**
Viena lieluma un viena burtveidola rakstzīmju kopa. Plašākā nozīmē — burtveidola konkrēta implementācija programmas formā.

657 **font card**
**fontkarte**
**шрифтовая карта**
Sk. *font cartridge*.

658 **font cartridge**
**fontu kasetne**
**кассетный шрифт**
Viena vai vairāku burtveidolu bitkartētu fontu vai kontūrfontu kopa, kas ierakstīti lasāmatmiņas modulī, ko var iespraust attiecīgajā printera ligzdā, tādējādi paplašinot printera fontu kopu.

659 **font editor**
**fontu redaktors**
**редактор шрифта**
Programmatūra, kas nodrošina lietotājus ar fontu modificēšanas un jaunu fontu veidošanas iespējām. Šādas programmas parasti izmanto fonta attēlus displeja ekrānā, kā arī attēlus, ko ielādē *PostScript* vai kādā citā printerī.

660 **font family**
**fontu saime**
**семья шрифтов**
Viena burtveidola dažāda lieluma fontu kopa, t.sk., piem., šo fontu slīpraksta un treknraksta variācijas.

661 **font generator**
**fontu ģenerators**
**генератор шрифта**
Programmatūra, kas pārveido rakstzīmju kontūrfontus noteikta stila un lieluma bitkartētajos fontos, kādi nepieciešami konkrētā dokumenta drukāšanai. Fontu ģeneratori mērogo rakstzīmju kontūru, bieži tie var arī izplest vai saspiest ģenerējamās rakstzīmes.

662 **font number**
**fonta numurs**
**номер шрифта**
Numurs, ar kura palīdzību lietojumprogramma vai operētājsistēma identificē noteiktu fontu.

663 **font page**
**fonta lappuse**
**страница шрифта**
Firmas *IBM* videostandarta *MCGA* sistēmās — videoatmiņas apgabals, kas tiek rezervēts programmētāju izveidotām rakstzīmju definīciju tabulām (rakstzīmju šablonu kopām), ko izmanto, lai attēlotu tekstu displeja ekrānā.

664 **font size**
**fonta lielums**
**размер шрифта**
Noteikta burtveidola rakstzīmju kopas punktu lielums.

665 **footer**
**kājene**
**нижний колонтитул**
Informācija, kas teksta apstrādes vai lappušu izkārtojuma programmās atkārtojas (lappuses numurs, dokumenta vai izdevuma nosaukums u.c.) un ko parasti drukā kāda izdevuma vai dokumenta lappuses beigās.

666 **footnote**
**vēre**
**сноска**
Tekstu apstrādes vai lappušu izkārtojuma procesā — zem lappuses teksta dota atsauce vai piezīme, uz kuru parasti ir norāde tekstā. Vairums teksta apstrādes programmu vēres automātiski numurē un atceras, ja tās tiek iespraustas tekstā vai svītrotas no tā. Sk. arī *endnote*, *footer*.

667 **For Your Information (FYI)**
***FYI* dokumenti**
**"к Вашему сведению"**
Apzīmējums dokumentiem, kas tiek izplatīti *RFC* dokumentu sērijas ietvaros un kas nav tieši saistīti ar tehniskiem

standartiem vai protokolu aprakstiem. *FYI* dokumentos tīkla *Internet* lietotājiem tiek sniegtas dažādas vispārējas ziņas, piem., par protokolu *TCP/IP* sistēmu, tīklu *Internet* vai izmantoto terminoloģiju.

**668 foreground**
**priekšplāns**
**передний план**

1. Prioritāte, kas vairākuzdevumu sistēmās tiek piešķirta interaktīvā režīmā strādājošajām programmām atšķirībā no pakešrežīmā strādājošajām programmām, kas ir fona darbi.
2. Kāda attēla priekšējā daļa.
3. Videoklipā — attēls, kas uzklāts uz cita attēla. Sk. arī *foreground colour*.

**669 foreground colour**
**priekšplāna krāsa**
**цвет символа**

Displeja ekrānā redzamā teksta vai grafikas krāsa, kas atšķiras no fona krāsas. Sk. arī *foreground*.

**670 form feed (FF)**
**formas padeve**
**перевод страницы**

Printera komanda, kas liek tam pamest pašreiz izvadāmo lappusi un pārvietoties uz nākošās lappuses augšmalu. Šo darbību veic vai nu nospiežot printera formas padeves pogu (*FF*) vai arī no datora, nosūtot printerim formas padeves rakstzīmi, kam koda *ASCII* decimālā vērtība ir 12.

**671 format**
**formāts**
**формат**

Noteikta informācijas izkārtojuma struktūra, ko izmanto informācijas apstrādei, ierakstīšanai datu vidē, attēlošanai displeja ekrānā vai izdrukai. Sk. arī *formatting*.

**672 formatting**
**formatēšana**
**форматирование**

1. Process, kura izpildes gaitā dokumentam tiek piešķirts noteikts ārējais veidols: malu un atkāpju lielums, burtstils un citi tā ārējā izskata elementi, kas nosaka dokumenta individualitāti un uzlabo tā lasāmību.
2. Datu vides (piem., magnētiskā diska, disketes) sagatavošana izmantošanai datu apstrādes procesā, ierakstot uz tās speciālu vadības informāciju.

**673 FoxPro**
**datu bāzes pārvaldības sistēma *FoxPro*, *FoxPro***
**система управления базой данных *FoxPro***

Firmas *Microsoft* izstrādāta datu bāzes pārvaldības sistēma, kas paredzēta firmas *IBM* un ar tiem saderīgajiem datoriem, kā arī *Macintosh* saimes personālajiem datoriem. *FoxPro* ir sistēmas *FoxBase* pilnveidojums, kas tai nodrošina spēju darboties operētājsistēmas *Microsoft Windows* vidē, izmantot valodas *SQL* un vaicājuma *QBE* saskarnes, kā arī lielās datu bāzēs ātri apkalpojamu vaicājumu tehnoloģiju.

**674 fractal**
**fraktālis**
**фрактал**

Attēlā ietverta ģeometriska figūra (kontūra), kas precīzi saglabā savu apveidu neatkarīgi no tā, kā tiek palielināts vai samazināts pats attēls (piem., krasta līnija, mākonis, koks u.c.).

675 **fractal image format (FIF)**
**fraktāļattēlu formāts, formāts *FIF*, *FIF***
**формат *FIF***
Datnes formāts tādu grafikas attēlu uzglabāšanai, kuri ir ievērojami saspiesti, izmantojot fraktāļus.

676 **fragmentation**
**fragmentēšana**
**фрагментация**
Operētājsistēmā *DOS* — datnes izkaisīšana pa dažādiem diskatmiņas apgabaliem, pievienojot datnei jaunus datus brīvajos atmiņas apgabalos.

677 **frame grabber**
**kadru tvērējs**
**устройство фиксации кадров**
Speciāla ierīce, kas paredzēta tieši no videokameras vai cita videoattēla avota saņemto grafisko attēlu ievietošanai datora atmiņā tā, lai tiktu nodrošināta to tālākā apstrāde. Kadru tvērējs saņem televīzijas standartsignālus un pārveido kārtējo videokadru ciparu formā, izveidojot datorā atbilstošu grafikas attēlu.

678 **freenet**
**brīvtīkls**
**сеть свободного доступа**
Organizācijas datoru tīkls, kas nodrošina bezmaksas vai ļoti lētu pieeju tīklam *Internet*. Brīvtīkli parasti tiek izveidoti publiskajās bibliotēkās vai universitātēs.

679 **freeware**
**brīvprogrammatūra**
**свободно распространяемые программы**
Datorprogrammas, ko lietotājam dod bez maksas. Šādas programmas izplata, paziņojot par to esamību lietotājiem. Izstrādātājs var aizliegt lietotājam šīs programmas izplatīt tālāk.

680 **frequency division multiplexing**
**frekvenčdales multipleksēšana**
**частотное мультиплексирование**
Multipleksēšanas paņēmiens, kas, izmantojot dažādas nesējfrekvences, nodrošina iespēju pa vienu sakaru kanālu pārraidīt vairākus savstarpēji neatkarīgus signālus (piem., audio- un videosignālus). Sk. arī *time division multiplexing*.

681 **frequency modulation (FM)**
**frekvences modulācija, *FM* tehnoloģija**
**частотная модуляция**
Informācijas kodēšana, mainot nesējfrekvenci. Ja šo tehnoloģiju izmanto ierakstīšanai magnētiskajā diskā, tad vienlaicīgi ar informācijas bitiem diskā tiek ierakstīti arī sinhronizācijas biti un diska virsma tiek izmantota neefektīvi. Ieraksta blīvuma paaugstināšanai izmanto *MFM* vai *RLL* tehnoloģiju.

682 **Frequently Asked Questions (FAQ)**
**saraksts "Bieži uzdodamie jautājumi", saraksts *FAQ*, *FAQ***
**часто задаваемые вопросы**
Kādas interešu grupas uzdodamo jautājumu saraksts. Šajā sarakstā ir arī atbildes uz jautājumiem, un tas parasti tiek noformēts kā dalīta ziņojumdēļa sistēma, piem., kā tīkls *USENET*, kuru var izmantot tīkla *Internet* lietotāji.

683 **FTP**
Sk. *File Transfer Protocol*.

**684 full motion video adapter**
**pilnkustības videoadapteris**
**кинематографический видеоадаптер**

Videoadapteris, kas nodrošina kustīgu videoattēlu izspīdināšanu datora displeja ekrāna logā. Lai izspīdinātu ekrānā videoattēlus, pilnkustības videoadapteris tiek savienots ar videokasešu vai optisko disku atskaņotāju vai ar videokameru (tiešraides gadījumā).

**685 full-screen application**
**pilnekrāna lietojums**
**полноэкранная прикладная программа**

Iespēja izmantot visu displeja ekrāna laukumu, izpildot lietojumprogrammas logošanas vidē. Izmantojot operētājsistēmu *Microsoft Windows*, logu var palielināt, noklikšķinot maksimizēšanas pogu, kas atrodas labajā virsrakstjoslas galā. Izmantojot operētājsistēmu *DOS*, logu palielina, nospiežot alternēšanas un ievadīšanas taustiņus.

**686 full-screen editor**
**pilnekrāna redaktors**
**экранный редактор**

Tekstu apstrādes programma, ko bieži iekļauj lietojumprogrammu izstrādāšanas sistēmās un kas paredzēta šo programmu izveidošanai. Sākot ar *MS-DOS* versiju 5.0 kvalitatīvs pilnekrāna redaktors (*MS-DOS Editor*) ir operētājsistēmas standartsastāvdaļa. Sk. arī *full-screen application*.

**687 function**
**funkcija**
**функция**

1. Mainīgs lielums, kura vērtība ir atkarīga no cita lieluma (lielumu) vērtības.
2. Kādas iekārtas, programmas vai sistēmas darbības uzdevums.
3. Programmēšanas valodās un izklājlapu programmās — procedūra, kuru izpildot tiek formēta kāda vērtība un kuras izsaukums var tikt izmantots kā kādas izteiksmes operands.

**688 function key**
**funkcijas taustiņš**
**функциональная клавиша**

Taustiņš, kura nospiešana iniciē no datorā izmantotās programmatūras atkarīgas speciālas funkcijas izpildi. Funkcijas taustiņus parasti apzīmē ar F1, F2, utt.

**689 functional decomposition**
**funkcionālā dekompozīcija**
**функциональная декомпозиция**

Dekompozīcija, kuras rezultātā sistēma tiek sadalīta komponentos (funkcionālos moduļos), kas izpilda noteiktas sistēmas funkciju apakškopas un kas kopumā veic visas sistēmas funkcijas.

**690 functional design**
**funkcionālā projektēšana**
**функциональное проектирование**

Sistēmas komponentu funkcionālo sakaru noteikšanas process.

**691 functional specification**
**funkcionālā specifikācija**
**функциональное описание**

Specifikācijas veids, kuru izmantojot katra sistēmas funkcionālā moduļa (ierīces, programmas) darbība tiek aprakstīta ieejas un izejas terminos, norādot, ko šie bloki dara, bet neatklājot, kā tie veic savas funkcijas.

692 **functional testing**
**funkcionālā testēšana**
**функциональное тестирование**
Testēšana, kas analizē tikai sistēmas atbildes reakciju uz noteiktām ieejas iedarbībām, bet ignorē tās iekšējo struktūru.

693 **FYI**
Sk. *For Your Information.*

## G

694 **game control adapter**
**spēļu vadības adapteris**
**игровой адаптер**
Īpaša adaptera karte ar pieslēgvietu kursorsviras vai spēļvadnes pievienošanai datoram, lai to varētu izmantot datorspēlēm. Sk. arī *game port.*

695 **game port**
**spēļu pieslēgvieta**
**игровой порт**
Pieslēgvieta kursorsviras vai spēļvadnes pievienošanai datoram. Sk. *game control adapter.*

696 **games paddle**
**spēļvadne**
**игровой пульт**
Rokā turama ierīce, ar kuras palīdzību datorspēlēs pārvieto kursoru vai attēlu.

697 **gas-discharge display**
**gāzizlādes displejs**
**газоразрядный дисплей**
Sk. *plasma display.*

698 **gas-plasma display**
**gāzplazmas displejs**
**плазменный дисплей**
Sk. *plasma display.*

699 **gateway**
**vārteja**
**шлюз**
Ierīce, kas nodrošina datu pārraidi starp datoru tīkliem, kuriem ir atšķirīga arhitektūra un protokoli (piem., lokālo datoru tīklu un pakešu komutācijas datoru tīklu). Vārteja parasti ir aparatūras un programmatūras apvienojums, kam ir savs procesors un atmiņa un kas veic protokolu pārveidošanu.

700 **Gb**
Sk. *gigabit.*

701 **GB**
Sk. *gigabyte.*

702 **GDI**
Sk. *Graphical Device Interface.*

703 **ghost**
**māņattēls**
**паразитное изображение**
Bāls, sekundārs attēls, kas parādās cieši blakus primārajam attēlam. Māņattēla parādīšanos parasti izsauc sekundārie signāli, kas pārraides laikā pienāk pirms vai pēc primārā signāla uztveršanas.

704 **GIF**
Sk. *graphics interface format.*

705 **giga- (G)**
**giga- (G)**
**гига-**
1. Prefikss, kas norāda uz kāda lieluma

skaitliskās vērtības reizinājumu ar 1 miljardu ($10^9$).
2. Prefikss, kas datortehnikā norāda uz kāda lieluma skaitliskās vērtības reizinājumu ar $2^{30}$ (1 073 741 824).

706 **gigabit (Gb)**
**gigabits (Gb)**
**гигабит**
Viens gigabits ir $2^{30}$ jeb 1 073 741 824 biti. Sk. arī *giga-*.

707 **gigabyte (GB)**
**gigabaits (GB)**
**гигабайт**
Viens gigabaits ir $2^{30}$ jeb 1 073 741 824 baiti. Sk. arī *giga-*.

708 **GKS**
Sk. *Graphics Kernel System*.

709 **glare filter**
**žilbumfiltrs**
**противобликовый фильтр**
Filtrs, ko izmanto, lai likvidētu spožus gaismas atstarojumus displeja ekrānā.

710 **glitch**
**klupe**
**дефект**
Aparatūras īslaicīga vai gadījumrakstura disfunkcija, ko var radīt kāda negaidīta signālu izmaiņa shēmā un kas savukārt var radīt kļūdainu datora izejas informāciju vai ekstremālā situācijā — sistēmas avāriju.

711 **glossary**
**glosārijs**
**глоссарий**
Bieži lietojamu frāžu un teksta fragmentu kopums, kas tiek veidots ar tekstapstrādes programmu starpniecību un ko vajadzības gadījumā var gatavā veidā iespraust sagatavojamajā dokumentā.

712 **Gopher**
**sistēma *Gopher***
**система *Gopher***
Tīkla *Internet* resursu un pakalpojumu atrašanas, piesaistīšanas un izmantošanas interaktīva sistēma, kas izstrādāta Minesotas universitātē ASV. Saskarne ar lietotājiem tiek nodrošināta, izmantojot dažādu izvēlņu kopu.

713 **Graphical Device Interface (GDI)**
**grafiskā ierīču saskarne, saskarne *GDI***
**интерфейс графического устройства**
Grafiskās lietotāja saskarnes daļa, kas ļauj programmētājam atveidot displeja ekrāna dialoglodziņus un citus grafikas elementus, izsaucot grafiskajā ierīču saskarnē šim nolūkam paredzētās procedūras.

714 **Graphical User Interface (GUI)**
**grafiskā lietotāja saskarne**
**графический интерфейс пользователя**
Displeja formatēšanas veids, kas ļauj lietotājam izvēlēties komandas, palaist programmas, kā arī apskatīt datņu sarakstus un citus objektus, norādot to piktogrāfiskos attēlus (ikonas). Izvēli var izdarīt, izmantojot tastatūru vai peli. Grafiskā lietotāja saskarne piedāvā vidi, kas nodrošina tiešu dialogu ar datoru. Grafisko lietotāja saskarni izmanto, piem., operētājsistēma *Microsoft Windows*.

715 **graphics accelerator**
**grafikas paātrinātājs**
**акселератор графики**
Paplašināšanas plate ar grafikas līdzprocesoru un ātrdarbīgu brīvpieejas atmiņu, kas paredzēta grafisko zīmēšanas operāciju paātrināšanai. Grafikas paātrinātāju parasti izmanto, lai uzlabotu lietotāja grafiskās saskarnes darbību vai arī lai paaugstinātu tādu grafikas piesātinātu lietojumu efektivitāti, kas saistīti ar multividi un datorizdevniecību.

716 **graphics adapter**
**grafikas adapteris**
**графический адаптер**
Videoadapteris, kas ļauj vizualizēt kā grafiku, tā arī tekstu. Personālajos datoros tas parasti izveidots kā izvēršanas plate, kas pārveido programmatūras komandas signālos, kuri attēlo grafiku un tekstus displeja ekrānā.

717 **graphics character**
**grafikas rakstzīme**
**графический символ**
Rakstzīme, kura apvienojumā ar citām rakstzīmēm var veidot vienkāršus grafiskus attēlus, piem., logus, ietonētus vai melnus četrstūrus, kā arī citas formas.

718 **graphics coprocessor**
**grafikas līdzprocesors**
**графический сопроцессор**
Specializēts mikroprocesors, kas, izpildot datora instrukcijas, ģenerē grafikas attēlus (līnijas, krāsainus laukumus u.c.), tādējādi atbrīvojot centrālo procesoru citiem uzdevumiem. Parasti grafikas līdzprocesors ir viena no videoadaptera sastāvdaļām.

719 **graphics device interface**
***grafisko ierīču saskarne***
**интерфейс графических устройств**
Operētājsistēmā *Microsoft Windows* — lietojumprogrammu funkciju kopa, kas nodrošina gan grafisko izvadu ekrānā, gan arī grafiskajam izvadam vajadzīgo ploteru un printeru atlasi. Sk. arī *interface*.

720 **graphics interface format (GIF)**
**grafikas interfeisa formāts, formāts *GIF*, *GIF***
**формат графического интерфейса**
Bitkartētas grafikas datnes formāts, ko parasti izmanto operatīvās informācijas sistēmās, jo tas nodrošina tādas augstas izšķirtspējas grafikas kā skenētu attēlu efektīvu saspiešanu.

721 **Graphics Kernel System (GKS)**
**grafikas kodola sistēma, standarts *GKS*, *GKS***
**базовая графическая система**
Amerikas Nacionālā standartu institūta un Starptautiskās standartizācijas organizācijas rekomendēts datorgrafikas standarts, kas nodrošina programmētājus ar standartizētām grafisko attēlu aprakstīšanas, apstrādāšanas, uzglabāšanas un pārraidīšanas metodēm. Grafikas kodola sistēmas izmantošana atvieglo grafisko lietojumprogrammu pārnešanu no vienas datu apstrādes sistēmas uz citu.

722 **graphics mode**
**grafikas režīms**
**графический режим**
Režīms, kurā displeja ekrāns tiek aplū-

kots kā pikseļu matrica. Rakstzīmes un citas kontūras tiek veidotas, izmantojot noteiktas pikseļu kombinācijas. Grafiskais režīms ir ērtāks lietošanai, jo programmas, kas nodrošina grafisko režīmu atšķirībā no teksta režīma programmām var attēlot praktiski neierobežotu formu un fontu dažādību.

**723 graphics primitive**
**grafikas primitīvs**
**графический примитив**

Grafikas pamatelements (loks, līnija, aizpildīts laukums u.c.), ko izmanto, lai veidotu sarežģītākus grafikas attēlus.

**724 graphics processor**
**grafikas procesors**
**графический процессор**

Specializēts mikroprocesors, kas parasti iebūvēts videoadapterī un kas atbilstoši datora instrukcijām veido grafiskos attēlus, tādējādi atbrīvojot datoru (centrālo procesoru) citiem darbiem.

**725 graphics spreadsheet**
**grafikas izklājprogramma**
**графическая электронная таблица**

Izklājlapu programma, kas atveido displeja ekrānā darblapu, izmantojot bitkartētu grafiku. Tādas grafikas izklājlapu programmas kā *Microsoft Excel* un *Lotus 1-2-3*, kas paredzētas darbam operētājsistēmas *Microsoft Windows* vidē, parasti ietver plašu datorizdevniecības elementu spektru un vienā izdrukā ļauj kombinēt izklājlapas un diagrammas.

**726 graphics terminal**
**grafiku terminālis**
**графический терминал**

Speciāls terminālis ar augstas izšķirtspējas displeju, grafikas planšeti vai citām grafikas ievadierīcēm, kas paredzēts grafisko datu ievadei personālajā datorā un izvadei no tā.

**727 gray scale**
**pelēkuma skala**
**полутоновая шкала**

Pelēko nokrāsu sērija, kas no balta pāriet melnā. Pelēko skalu lieto grafiskajos attēlos, lai tiem pievienotu nepieciešamās detaļas. Pelēko toņu skaits ir atkarīgs no bitu skaita, ko izmanto katra attēla pikseļa "krāsas" aprakstam — jo vairāk bitu izmanto pikseļa aprakstam, jo vairāk gradāciju iegūst. Pieaugot kodējošo bitu skaitam, strauji pieaug tiem nepieciešamās atmiņas apjoms.

**728 greeking**
**imitējumraksts**
**имитирование [текста]**

Patvaļīgu burtu vai simbolu izmantošana, lai parādītu drukājamās lappuses izskatu, neparādot faktiski drukājamo tekstu. Imitējumrakstu izmanto gadījumos, kad publikācijai paredzētajā lappusē teksts ir pārāk sīks, lai to varētu ekrānā izlasīt.

**729 green PC**
**zaļais personālais dators, zaļais *PC***
**зеленый персональный компьютер**

Datoru sistēma, kas izstrādāta, lai varētu darboties energoekonomijas režīmā. Atšķirībā no standartsistēmām jaudīgi zaļie *PC* patērē apmēram 1,5 reizes mazāk elektroenerģijas. Zaļajos *PC* ir paredzēti speciāli energotaupīšanas režīmi, kuros

tiek samazināts monitora ekrāna spilgtums un apturēta cietā diska rotācija. Miega režīmā šie datori parasti patērē tikai 28-36 W.

730 **grid**
**režģis**
**сетка**

Divas līniju kopas, kuru elementi krustojas taisnā leņķī, piem., izklājlapa ir rindu un kolonu režģis, displeja ekrāns — horizontālu un vertikālu punktu (pikseļu) režģis. Režģis atvieglo paralēlu un perpendikulāru līniju, kā arī diagramu zīmēšanu.

731 **GUI**

Sk. *Graphical User Interface.*

732 **hacker**
**urķis**
**хакер**

Datoru izmantošanas entuziasts, kam sagādā prieku izpētīt dažādas datoru sistēmas un atrast pieeju tajās uzglabātajai informācijai.

733 **half height drive**
**pusaugstais diskdzinis**
**полувысокий дисковод**

Diskdzinis, kura augstums ir puse no sākotnējā firmas *IBM* personālo datoru diskdziņu augstuma, kas bija 3 collas. Pusaugstais diskdzinis ir mūsdienu personālo datoru standartelements.

734 **halftoning**
**pustonēšana**
**формирование полутоновых изображений**

Mainīgas gradācijas toņu attēlu drukāšanas tehnoloģija, izmantojot dažāda lieluma punktus, kuri, saplūstot kopā, veido dažādas attēla elementu pelēkuma pakāpes. Sk. arī *dithering, gray scale.*

735 **handle**
**turis**
**зацепка**

1. Datorgrafikas un datorizdevniecības programmās — mazi melni laukumi, kas parasti izvietoti ap katru objektu. Velkot aiz tura, ar peles palīdzību var mainīt izvēlētā objekta novietojumu vai tā izmērus.
2. Lokāls identifikators, ar kura palīdzību var realizēt pieeju kādai ierīcei vai tādiem objektiem kā dators, logi vai dialoglodziņi. Grafiskajā lietotāja saskarnē turus izmanto šo objektu viennozīmīgai identificēšanai.

736 **handwriting recognition**
**rokraksta pazīšana**
**распознавание рукописного текста**

Datora spēja pazīt ar roku rakstītu tekstu un pārveidot to koda *ASCII* vai citās rakstzīmēs.

737 **hard card**
**cietā karte**
**аппаратурная карта**

Cietā diska analogs, kas kopā ar vadības shēmu izveidots kā datora slotā ievietojama karte. Cietās kartes atmiņas ietilpība ir mazāka nekā cietajam diskam, bet tā viegli pievienojama datoram un tās darbības ātrums ir lielāks nekā parastajiem diskiem.

738 **hard copy**
**cietā kopija**
**твердая копия**

Paliekoša no personālā datora iegūstamas informācijas izvades forma, piem., izdruka, filma, dokuments. Sk. arī *soft copy*.

739 **hard disk**
**cietais disks**
**жесткий диск**
Personālā datora lielas ietilpības palīgatmiņa, kas sastāv no vairākiem magnētiskajiem diskiem, to piedziņas iekārtas (diskdziņa), lasīšanas/rakstīšanas galviņu komplekta un elektroniskas saskarnes, kas nodrošina šīs atmiņas iekārtas sadarbību ar personālo datoru. Sk. arī *Entranced Small Device Interface*, *Integrated Drive Electronics*, *Small Computer System Interface*, *ST 506*.

740 **hard return**
**stingrā atgrieze**
**твердый возврат**
Vadības kods, kas liek kursoram vai printerim pāriet uz jaunu rindiņu. Teksta apstrādes programmās, kas automātiski kārto rindiņas lappusē, stingro atgriezi lieto tikai rindkopas beigās. To ievada dokumenta tekstā, nospiežot ievadtaustiņu.

741 **hardware**
**aparatūra**
**аппаратура**
Datu apstrādes sistēmas fizikālā daļa, kurā ietilpst elektriskās, elektroniskās un elektromehāniskās shēmas, iekārtas un to savienojumi (t.sk. dažādas ievadizvades ierīces, rādītājierīces u.c.), kā arī konstruktīvie elementi (piem., statnes).

742 **hardware cache**
**aparatūras kešatmiņa**
**аппаратная кэш-память**
Diskdzinī vai diskdziņa kontollerī izveidota kešatmiņa, kurā tiek saglabātas bieži lietojamas programmas un dati. Aparatūras kešatmiņā ierakstītajiem datiem dators var piekļūt daudz ātrāk, nekā datiem, kas glabājas diskatmiņā. Sk. arī *primary cache*, *secondary cache*, *software cache*.

743 **hardware interrupt**
**aparatūrpārtraukums**
**аппаратное прерывание**
Pārtraukums, ko rada vai nu ārējo iekārtu, kā, piem., tastatūras, diskdziņu, ievadizvades pieslēgvietu, peles, vai arī centrālā procesora darbība.

744 **hardware key**
**aparatūras atslēga**
**аппаратурный ключ**
Īpašs slēdzis vai speciāla ierīce, kas aizsargā datoru vai noteiktu programmatūru pret nesakcionētu izmantošanu. Sk. arī *copy protection*.

745 **head parking**
**galviņievilkšana**
**позиционирование головок**
Cietā diska galviņu fiksēšana pozīcijā, kas izslēdz plates virsmas bojāšanos, diskdzini pārvietojot. Vairumam moderno diskdziņu, izslēdzot barošanas spriegumu, galviņas fiksējas automātiski.

746 **header**
**galvene; iesākums**
**заголовок**
1. Tekstu apstrādē — viena vai vairākas identificējošas rindiņas, ko drukā lappuses augšā. Galvenē parasti ir lappuses numurs, datums, autoru vārdi vai dokumenta virsraksts. Sk. arī *footer*.

2. Datu apstrādē — pirmais ieraksts, kas identificē datni. Galvene var saturēt datnes nosaukumu, tās pēdējo izmaiņu datumu un citu informāciju.
3. Datu pārraidē — ziņojuma sākuma daļa, kurā ir vadības informācija, ziņas par datus pārraidošo un uztverošo staciju, ziņojuma tipu un tā prioritātes līmeni. Sk. arī *trailer*.

### 747 help palīdzība помощь

Interaktīvās sistēmas daļa, kas ļauj lietotājam pēc pieprasījuma iegūt informāciju par sistēmas darbību, lai izvēlētos vajadzīgo ceļu sava uzdevuma risināšanai.

### 748 helper program līdzētājprogramma программа-помощник

Papildprogramma globālā tīmekļa *WWW* pārlūkprogrammā. Tā pārlūkprogrammai ļauj apstrādāt multivides datnes, kurās ir, piem., videoinformācija, audioinformācija un informācija par kustīgiem attēliem. Vairums līdzētājprogrammu pieder pie brīvprogrammatūras vai izplatāmprogrammatūras.

### 749 Hercules Graphics videostandarts *Hercules Graphics*; videoadapteris *Hercules Graphics* видеостандарт *Hercules Graphics*; видеоадаптер *Hercules Graphics*

1. Videostandarts, kas saderīgs ar videostandartu *MDA* un dod iespēju strādāt ne tikai vienkrāsas teksta, bet arī grafikas režīmā ar izšķirtspēju 720×348.
2. Videoadapteris, kas atbilst videostandartam *Hercules Graphics* un, izmantojot relatīvi lētu vienkrāsas monitoru, nodrošina iespēju realizēt grafikas režīmu.

### 750 Hercules Graphics Adapter

Sk. *Hercules Graphics*.

### 751 Hercules Graphics Card (HGC) *Hercules Graphics* karte графическая карта *Hercules Graphics*

Augstas izšķirtspējas grafikas karte (videoadapteris) monohromiem monitoriem, ko izstrādājusi firma *Hercules Computer Technology*. Sk. arī *Hercules Graphics*.

### 752 Hewlett-Packard Company (HP) firma *Hewlett-Packard Company*, firma *HP* фирма *Hewlett-Packard Company*

Viena no lielākajām ASV firmām, kas specializējusies elektrisko mēraparātu, kalkulatoru, minidatoru un personālo datoru, kā arī to ārējo iekārtu izstrādē un ražošanā. Savu darbību personālo datoru jomā firma *HP* uzsāka ar personālā datora *Touchscreen 150*, kas darbojās operētājsistēmas *MS-DOS* vadībā, ražošanu. 1984. g. firma sāka ražot *IBM PC* saderīgu datoru saimi *Vectra*, kuros sākumā izmantoja mikroprocesoru *Intel 80286*, bet vēlāk mikroprocesoru *Intel 80486* un *Intel Pentium*. 1986. g. firma *HP* izstrādāja *RISC* datoru arhitektūru — *HP Precision Architecture*, ko izmantoja jaunāko minidatoru *HP 3000* un *UNIX* darbstaciju *HP 9000* modeļu izveidei. Pēc apvienošanās ar firmu *Appollo Computer* firma *HP* kļuva par vienu no

lielākajām darbstaciju ražotājām pasulē. Šī firma izstrādājusi un ražo strūklprinterus *HP LaserJet*, kā arī *HP NetServer* saimes datoru tīklu serverus.

**753 HFS**
Sk. *hierarchical file system*.

**754 HGC**
Sk. *Hercules Graphics Card*.

**755 hidden attribute**
**slēptais atribūts**
**скрытый атрибут**
Operētājsistēmās *DOS* un *OS/2* — datnes atribūts, kas norāda, ka informācija par datni neparādās parastajās direktoriju izdrukās. Sk. arī *hidden file*.

**756 hidden file**
**slēptā datne**
**скрытый файл**
Datne, ar apslēptu atribūtu kopu, kas norāda operētājsistēmai, ka informācija par šo datni neparādās parastajās direktoriju izdrukās. Slēptās datnes parasti nevar dzēst, kopēt vai arī izspīdināt to saturu displeja ekrānā.

**757 hierarchical decomposition**
**hierarhiskā dekompozīcija**
**иерархическая декомпозиция**
Dekompozīcija, kuras rezultātā sistēma tiek sadalīta komponentu hierarhijā, ievērojot lejupejošus precizējumus.

**758 hierarchical file system**
**hierarhiska datņu sistēma, sistēma *HFS***
**иерархическая система файлов**
Datņu organizēšanas metode, kas operētājsistēmās *MS-DOS* un *OS/2* izmanto speciālu datņu hierarhiju, ko sauc par direktorijiem vai mapēm (personālo datoru saimē *Macintosh*). Hierarhiskā datņu sistēma sākas ar saknes direktoriju, no kā atzarojas apakšdirektoriji. Katrā direktorijā vai apakšdirektorijā var būt gan datnes, gan citi direktoriji.

**759 high density (HD) disk**
**lielblīvuma diskete, *HD* diskete**
**дискета высокой плотности**
Augstas kvalitātes diskete, kurā var uzglabāt ievērojami lielāku datu daudzumu nekā dubultblīvuma disketē (*DD*). Piem., 3,5 collu *HD* disketes kapacitāte ir 1,44 megabaiti, bet *DD* disketes — tikai 720 kilobaiti.

**760 high memory**
**augšējā atmiņa**
**верхняя память**
Personālajos datoros, kuru pamatatmiņa ir 1 megabaits, augšējā atmiņa ir pamatatmiņas daļa starp 640 kilobaitiem un 1 megabaitu. Pirmos 640 kilobaitus parasti izmanto kā brīvpieejas atmiņu, bet augšējo atmiņu rezervē dažādas aparatūras (piem., videoadapteru, seriālo pieslēgvietu u.c.) vadībai un lasāmatmiņas ievadizvades pamatsistēmai. Sk. arī *low memory*.

**761 high memory area (HMA)**
**augšējās atmiņas apgabals, atmiņas apgabals *HMA***
**область верхней памяти**
Personālajos datoros, kuru pamatatmiņas apjoms ir 1 megabaits, augšējās atmiņas apgabals ir pirmie paplašinātās atmiņas 64 kilobaiti (no 1024 kilobaitiem līdz 1088 kilobaitiem). Sākot ar versiju 5.0, operētājsistēmā *MS-DOS* ir iekļauts drai-

veris *HIMEM.SYS*, kas dod iespēju lietotājam izmantot šo atmiņas apgabalu operētājsistēmas programmu izvietošanai, tā palielinot lietojumprogrammu izmantojamo atmiņas apjomu.

762 **highlighting**
**izgaismošana**
**высвечивание**
Attēla detaļu vai segmenta izcelšana displeja ekrānā, mainot tā vizuālās īpašības, piem., gaišumu, mirgošanas režīmu un krāsas. Sk. arī *reverse video*.

763 **HIMEM.SYS**
**draiveris *HIMEM.SYS***
**драйвер *HIMEM.SYS***
Specifikācijā *XMS* paredzēts draiveris, kas programmām, kuras darbojas operētājsistēmu *DOS* un *Microsoft Windows* vidē, nodrošina iespēju izmantot paplašināto atmiņu.

764 **hit**
**trāpījums**
**совпадение**
1. Ieraksts, kurš apmierina specifisku meklēšanas kritēriju kopu datu bāzē.
2. Savienojums, kas izveidots ar *Web* serveri.

765 **HMA**
Sk. *high memory area*.

766 **home brew**
**mājdarinājums**
***любительское изделие***
Aparatūra, ko mājas apstākļos (dažkārt garāžā vai pagrabtelpās) izveidojuši datortehnikas entuziasti.

767 **home computer**
**sadzīves dators**
**бытовой компьютер**
Personālais dators, kas paredzēts dažādu skaitļošanas un informācijas apstrādes uzdevumu risināšanai sadzīves vajadzībām, kā arī datorspēlēm.

768 **Home key**
**sākumvietas taustiņš**
**клавиша возврата в исходное положение**
Taustiņš, kas apvienots ar taustiņu 7 un izvietots firmas *IBM* un ar to saderīgas tastatūras cipartastatūras daļā, kā arī paplašinātās tastatūras rediģēšanas taustiņu blokā starp pamattastatūru un cipartastatūru. Sākumvietas taustiņam dažādās programmās mēdz būt atšķirīgas funkcijas. Visbiežāk tas pārvieto kursoru uz rindiņas sākumu, uz kreiso augšējo ekrāna stūri vai uz dokumenta sākumu.

769 **home page**
**sākumlapa**
**начальная страница**
Globālā tīmekļa *WWW* dokuments, kas tiek uzskatīts par ieiešanas punktu kādā saistītu dokumentu kopā. Tā, piem., kādas firmas sākumlapā parasti ir firmas logotips, par šo firmu tīmeklī *WWW* sniegtās informācijas apraksts un saites ar dokumentiem, kas sniedz informāciju par šo firmu un ir pieejami tīmeklī *WWW*.

770 **hook**
**aizķere**
**зацепка**
Programmatūrā vai aparatūrā paredzēta iespēja pievienot tai specifiskas funkcijas lietotāja vajadzībām. Tā, piem., vārdu procesorā *Microsoft Word* ir iestrādāta iespēja izveidot dialoglodziņus, kas paplašina programmas izmantošanas iespējas speciāliem lietojumiem. Aparatūra

ar atvērtu arhitektūru, savukārt, atvieglo, piem., specializētu pārraudzības līdzekļu izveidošanu.

**771 host**
**resursdators**
**хост-машина**

Datoru tīkla centrālais vai vadošais dators, kas nodrošina nepieciešamos pakalpojumus citiem šī tīkla datoriem vai termināļiem. Arī tīklam *Internet* pievienotie datori, kas darbojas kā serveri, pilda resursdatoru funkcijas. Pieeju tiem nodrošina, izmantojot, piem., datņu pārsūtīšanas protokolu, sistēmu *Gopher* vai *WWW* pārlūkprogrammas.

**772 host adapter**
**hostadapteris**
**основной адаптер**

Adapteris, kas pārsūta datus un instrukcijas starp diskešu vai diskdziņa kontrolleri un centrālo procesoru. Parasti hostadapteris ir pievienots izvēršanas kopnei atbilstoši saskarnēm *IDE*, *EIDE* vai *SCSI*.

**773 hot key**
**karstais taustiņš**
**горячая клавиша**

Taustiņš vai taustiņu kombinācija, ko nospiežot tiek izraisīta kādas darbības izpilde neatkarīgi no tā, kāda programma tajā brīdī tiek datorā izpildīta.

**774 hot link**
**karstā saite**
**горячее соединение**

Tāda saikne starp diviem lietojumiem, kad mainot vienu lietojumu mainās arī otrs lietojums. Karsto saiti var nodibināt, piem., starp izklājlapu un noteiktu dokumentu, lai izklājlapā izdarītās maiņas izsauktu atbilstošas maiņas attiecīgajās dokumenta ailēs.

**775 hot spot**
**karstvieta**
**область активизации**

Vieta dokumentā, kurā atrodas iegultā hipersaite.

**776 hotlist**
**karstais saraksts**
**активный список**

Bieži izmantojamo unificēto resursu vietrāžu saraksts.

**777 HP**

Sk. *Hewlett-Packard Company*.

**778 HTML**

Sk. *Hypertext Markup Language*.

**779 HTTP**

Sk. *Hyper Text Transport Protocol*.

**780 hub**
**centrmezgls**
**концентратор**

Datu pārraides tīkla komunikācijas ierīce, kas modificē pārraidāmos signālus un ļauj pieslēgt datoru tīklam papildu darbstacijas. Centrmezgli kļuvuši pazīstami sakarā ar lokālo tīklu *ARCnet* arhitektūru un dažādu datu pārraides tīklu (t.sk. arī optisko) plašu izplatību. Dažos zvaigžņtīklos centrmezgls izpilda arī centrālās vadības ierīces funkcijas. Sk. arī *active hub*, *passive hub*.

**781 human-computer interface**
**cilvēka-datora saskarne**
**человеко-машинный интерфейс**

Saskarne, kas nodrošina lietotāja un

datora sadarbības uzlabošanu. Sk. arī *graphical user interface.*

**782 Hyper Text Transport Protocol (HTTP)**
**hiperteksta transporta protokols, protokols *HTTP***
**гипертекстовый транспортный протокол**

Tīkla *Internet* standartprotokols, kas nodrošina informācijas apmaiņu globālajā tīmeklī *WWW*. Protokolu *HTTP* izmanto hipersaišu veidošanai starp hipertekstа dokumentiem. Noklikšķinot peli uz kādas no hipersaitēm, ar šī protokola starpniecību tiek atvērts attiecīgais dokuments neatkarīgi no tā, kur šis dokuments tīklā *Internet* ir izvietots. Sk. arī *home page*, *Hypertext Markup Language*, *Standard Generalized Markup Language*, *Uniform Resource Locater.*

**783 hyperlink**
**hipersaite**
**гиперсвязь**

Hipertekstu sistēmās pasvītrots vai kā citādi izcelts vārds vai frāze, uz kura novietojot kursoru un noklikšķinot peli, displeja ekrānā tiek parādīts kāds cits dokuments. Hipersaiti bieži dēvē arī par enkuru.

**784 hypermedia**
**hipervide**
**гиперсреда**

Datu, teksta, grafikas un audioinformācijas elementu integrācija hipertekstu sistēmā. Hipervidi veido hipersaites, kas atvieglo lietotājam pāreju no vienas informācijas formas uz citu.

**785 hypertext**
**hiperteksts**
**гипертекст**

Datorizētas informācijas nelineāra pārlūkošanas un izguves sistēma, kas lietotājam ar īpaši izveidotu asociatīvu saišu (hipersaišu) palīdzību ļauj veikt informācijas izguvi no savstarpēji saistītiem dokumentiem neatkarīgi no to izvietojuma datora (datoru) atmiņā. Sk. arī *World Wide Web.*

**786 Hypertext Markup Language (HTML)**
**hiperteksta iezīmēšanas valoda, valoda *HTML*, *HTML***
**язык *HTML***

Valoda, kas, izmantojot speciālus kodus, nosaka hipertekstа dokumenta atveidojumu displeja ekrānā gadījumā, ja tiek lietotas tīkla *Internet* *WWW* lappuses (piem., kods "<p>" apzīmē jaunu rindkopu, bet kods "<b>" — treknrakstu displeja ekrānā). Sk. arī *Standard Generalized Markup Language.*

**787 hyphenation**
**zilbjdale**
**разбивка слов по слогам**

Tekstu apstrādes un izklājlapu programmās — operācija, kas automātiski sadala vārdu zilbēs, ja tas sniedzas pāri labajai dokumenta malai un pārnes daļu no tā uz nākošo teksta rindiņu.

**788 i 486**

Sk. *Intel 80486.*

789 **I/O**
Sk. *input/output.*

790 **IBM**
Sk. *International Business Machines Corporation.*

791 **IBM BIO.COM**
Sk. *IO.SYS.*

791 **IBM compatible computer**
***IBM* saderīgs dators**
**компьютер, совместимый с компьютерами фирмы *IBM***
Personālais dators, kas var strādāt ar visu vai gandrīz visu firmas *IBM* personālajiem datoriem izstrādāto programmatūru un kam izmantojamas firmas *IBM* personālo datoru kartes, adapteri un ārējās ierīces. Sk. arī *clone.*

793 **IBM compatible PC**
***IBM* saderīgs personālais dators**
***IBM* совместимый персональный компьютер**
Vispārīgs apzīmējums tādiem personālajiem datoriem, kas ir aparatūrsaderīgi un programmsaderīgi ar *IBM PC.*

794 **IBM DOS.COM**
Sk. *MSDOS.SYS.*

795 **IBM PC**
***IBM PC***
***IBM PC***
Firmas *IBM* personālo datoru apzīmējums, ko izmantoja pirmā šīs firmas 1981. g. izstrādātā datora apzīmēšanai un ar ko vēlāk apzīmēja visu firmas *IBM* personālo datoru saimi.

796 **IBM Personal System/2**
Sk. *IBM PS/2*

797 **IBM PS/2 (IBM Personal System/2)**
***IBM PS/2***
***IBM PS/2***
Apzīmējums firmas *IBM* personālo datoru saimei, kas izstrādāta 1987. g. kā agrāko šīs firmas personālo datoru pilnveidojums. Šajos datoros izmanto 3,5" lokanos diskus, adapteri *VGA*, arhitektūras *MCA* kopni un mikroprocesorus *Intel 80286*, *80368* un *80486*.

798 **IBM ThinkPad**
**piezīmjdators *IBM ThinkPad*, *IBM ThinkPad***
**блокнотный компьютер *IBM ThinkPad***
Jaunāko un populārāko firmas *IBM* piezīmjdatoru saime.

799 **icon**
**ikona**
**пиктограмма**
Ilustratīvs objekta vai funkcijas attēlojums displeja ekrānā. Ikona var attēlot gan objektus (datnes, programmas, dokumentus u.c.), ar ko lietotājs strādā, gan darbības, ko lietotājs var veikt. Izmantojot ikonas, lietotājam nav nepieciešams atcerēties komandas un ievadīt tās no tastatūras. Ikonas dod iespēju ar peli norādīt izvēlēto objektu vai izpildāmo darbību un ir viens no grafiskā lietotāja saskarnes "draudzīguma" faktoriem.

800 **IDE**
Sk. *Integrated Drive Electronics.*

801 **identifier**
**identifikators**
**идентификатор**
Rakstzīmju virkne, ko izmanto kādas funkcijas, procedūras vai mainīgā identificēšanai programmā, kā arī cietā diska vai disketes apzīmēšanai. Datu bāzēs identifikators viennozīmīgi raksturo ierakstā glabājamo informāciju.

802 **IEEE**
Sk. *Institute of Electrical and Electronics Engineers.*

803 **IEEE 802 Standards**
**standarti *IEEE 802*,**
**standarti *802***
**стандарты *IEEE 802***
Standartu sērija, kuru izstrādi koordinē Elektrotehnikas un elektronikas inženieru institūts. 802. sērija ietver standartu tiltiem starp dažādiem lokālajiem tīkliem (802.1), standartu datu posma slāņa vadībai (802.2), standartu daudzpieejas lokālajiem tīkliem ar nesēju jušanu un sadursmju atklāšanu (802.3), standartu marķiermaģistrāles lokālajiem tīkliem (802.4), standartu marķiergredzena lokālajiem tīkliem (802.5), standartu pilsēttīkliem (802.6), kā arī rekomendācijas platjoslas tehnikas izmantošanai (802.7), rekomendācijas optisko šķiedru tehnoloģijas izmantošanai (802.8), rekomendācijas datu, audio- un videoinformācijas integrētai izmantošanai lokālajos tīklos un integrēto pakalpojumu cipartīklos, kas atbilst standartam 802.9, kā arī tīkla drošības modeļa standarta 802.10, bezvada tīklu standarta 802.11 un ātrdarbīga tīkla *Ethernet* (100 megabiti sekundē) standarta 802.12 izstrādi.

804 **IGES**
Sk. *Initial Graphics Exchange Specification.*

805 **illegal character**
**aizliegtā rakstzīme**
**запрещенный знак**
Rakstzīme, kuras izmantošana saskaņā ar komandvadāmu programmu un programmēšanas valodu sintakses likumiem nav atļauta. Šīs rakstzīmes parasti tiek rezervētas speciālu funkciju apzīmēšanai un tās nevar izmantot, piem., datņu vārdu veidošanai.

806 **image compression**
**attēlu saspiešana**
**сжатие изображения**
Speciāls datu saspiešanas paņēmiens, ar kuru samazina grafikas datņu lielumu, kas parasti aizņem ievērojamu diskatmiņas apjomu.

807 **image processing**
**attēlu apstrāde**
**обработка изображения**
Datorizēta grafikas attēlu (fotogrāfiju, zīmējumu, videoattēlu u.c.) analīze, pārveidošana, uzglabāšana un izspīdināšana displeja ekrānā. Attēlu apstrāde parasti saistīta ar to kvalitātes uzlabošanu, un tā tiek plaši izmantota televīzijā, kinematogrāfijā, lasāmatmiņas kompaktdisku tehnoloģijā, medicīnā, metroloģijā un citās nozarēs.

808 **imagesetter**
**attēllicis**
**преобразователь**
**изображения**
Speciāla drukas ierīce, ar kuras starpniecību uz gaismas jutīga materiāla

(fotopapīra, filmas) iespējams izvadīt no datora augstas izšķirtspējas attēlus.

**809 imaging**
**attēlveidošana**
**отображение**

Attēlu elektroniska atveidošana datorā, izmantojot skenēšanu.

**810 import**
**imports**
**импорт**

Citas programmas vai datu apstrādes sistēmas datnes formāta pārvēršana formātā, ko var izmantot aplūkojamā programma vai sistēma. Programmai vai sistēmai, kas saņem importētos datus, jāprot tos interpretēt.

**811 in-line image**
**ierindots attēls**
**встроенное изображение**

Hiperteksta iezīmēšanas valodā sastādītā dokumentā iebūvēta grafikas daļa, ko ar pārlūkprogrammas palīdzību izspīdina displeja ekrānā vienlaicīgi ar šo dokumentu.

**812 inactive window**
**neaktīvais logs**
**неактивное окно**

Logu vidē, kas spēj vienlaicīgi atveidot uz displeja ekrāna vairākus logus, neaktīvais logs ir logs, ar kuru lietotājs pašreizējā brīdī nestrādā. Neaktīvo logu var pilnīgi vai daļēji pārklāt cits logs un tas nevar saņemt datus vai komandas no tastatūras.

**813 indent**
**atkāpe**
**отступ**

Telpa, kas paliek tukša, kad kāda rindiņa vai teksta bloks tiek nobīdīts attiecībā pret lappuses malu vai pārējo tekstu. Atkāpi izmanto, lai iezīmētu jaunu rindkopu vai izceltu kādu teksta daļu.

**814 index**
**indekss; priekšmetu rādītājs**
**индекс; предметный указатель**

1. Programēšanā — vesels skaitlis, kas nosaka kādas datu vienības vietu attiecībā pret citu datu vienību.
2. Dokumenta vai datnes sastāvā ietilpstošo elementu saraksts ar atsaucēm, kas nosaka elementa atrašanās vietu.
3. Datu bāzu vadības programmās — datne ar norādēm, kas nosaka datu bāzes ierakstu fizisko atrašanās vietu. Kārtojot datu bāzi vai meklējot tajā ierakstus, programma darbojas tieši ar šo norāžu datni.

**815 Industry Standard Architecture (ISA)**
**industriālā standartarhitektūra, arhitektūra *ISA***
**архитектура *ISA***

Arhitektūra, ko firma *IBM* izmantoja savu pirmo personālo datoru *PC/XT* un *PC/AT* kopņu izstrādāšanai. Dažkārt šīs kopnes sauc arī par *ISA* kopnēm.

**816 information message**
**informatīvais ziņojums**
**информационное сообщение**

Ziņojums, kas informē lietotāju, ka tam vajadzīgā darbība ir negaidīti pārtraukta un nevar tikt turpināta. Lietotājam jāapstiprina, ka šī darbība jāaptur vai arī

jāgriežas pēc palīdzības. Sk. arī *action message, warning message.*

817 **initial base font**
**sākotnējais pamatfonts**
**начальный базовый шрифт**

Fonts, ko tekstu apstrādes programmas automātiski izmanto visu dokumentu veidošanai. Cita fonta izvēle jāizdara lietotājam, izmantojot, piem., dokumentu izvēlni vai pamatfontu izvēlni.

818 **Initial Graphics Exchange Specification (IGES)**
**sākotnējā grafikas apmaiņas specifikācija, specifikācija *IGES***
**спецификация начального графического обмена**

Standartizēts datorgrafikas datņu formāts, ko ieteicis Amerikas Nacionālais standartu institūts un kas īpaši piemērots datorizētās projektēšanas programmu veidoto modeļu aprakstīšanai.

819 **ink jet printer**
**strūklprinteris**
**струйное печатающее устройство**

Printeris, kas rakstzīmju attēlus uz papīra atveido, no kapilārām sprauslām izsmidzinot krāsvielas strūklu. Šo sprauslu skaits jaunākajiem strūklprinteriem var sasniegt 60. Šāds sprauslu skaits nodrošina augstu attēla izšķirtspēju.

820 **input**
**ievade**
**ввод**

Informācija, ko apstrādei datorā ievada ar ievadierīču starpniecību vai arī no datnēm, kas glabājas datora diskatmiņā.

821 **input device**
**ievadierīce**
**устройство ввода**

Ierīce, kas pārsūta datus, programmas vai vadības signālus datora procesoram un kalpo par saskarni starp cilvēku un datoru. Sk. arī *digitizer, joystick, keyboard, light pen, modem, mouse, scanner, trackball.*

822 **input/output (I/O)**
**ievadizvade**
**ввод-вывод**

Kopējs nosaukums ievades un izvades procesiem, kā arī attiecīgajiem datiem un aparatūrai, ko izmanto cilvēka un datora sadarbības nodrošināšanai. Šaurākā nozīmē — datu pārsūtīšanas process datorā (piem., starp procesoru un ārējām iekārtām), kura izpildes gaitā viena funkcionālā bloka izvaddati tiek iesūtīti cita bloka ievadierīcē.

823 **Ins key**

Sk. *Insert key.*

824 **Insert key**
**iespraušanas taustiņš**
**клавиша вставки**

Taustiņš, kas apvienots ar taustiņu 0 un izvietots firmas *IBM* un ar to saderīgas tastatūras cipartastatūras daļā, kā arī paplašinātās tastatūras rediģēšanas taustiņu blokā starp tastatūras pamatdaļu un cipartastatūru. Iespraušanas taustiņam dažādās programmās mēdz būt atšķirīgas funkcijas. Visbiežāk tas pārslēdz programmu no iespraušanas režīma uz pārrakstīšanas režīmu vai otrādi.

825 **insert mode**
**iespraušanas režīms**
**режим вставки**

Programmas darbības režīms, kurā teksts tiek ierakstīts dokumentā vai komandu rindā, vienlaikus pārbīdot pa labi visas rakstzīmes, kas ir aiz kursora, nevis pārrakstot tekstu pāri esošajam. Iespraušanas režīmam pretējs ir pārrakstīšanas režīms.

826 **insertion point**
**iespraušanas punkts**
**точка вставки**
Personālo datoru saimes *Macintosh* datoros izmantojamo operētājsistēmu, kā arī operētājsistēmas *Microsoft Windows* lietojumos — vertikāla mirgojoša svītra, kas norāda, kurā vietā uz ekrāna parādīsies ievadāmais teksts.

827 **installation program**
**instalēšanas programma**
**программа установки**
Programma, kura sagatavo citu programmatūras pakotni darbam datorā. Tā pārkopē datnes no distributīvās disketes uz cietā diska un identificē datora ārējās iekārtas, lai noteiktus programmu draiverus piekārtotu konkrētiem displejiem, printeriem, skeneriem un citām ierīcēm.

828 **Institute of Electrical and Electronics Engineers (IEEE)**
**Elektrotehnikas un elektronikas inženieru institūts, institūts *IEEE***
**Институт инженеров по электротехнике и радиоэлектронике (ИИЭР)**
Starptautiska profesionāla organizācija, kas apvieno inženierus, zinātniekus un studentus elektronikas un tai radniecīgās nozarēs. Viens no šīs organizācijas galvenajiem darbības virzieniem ir standartizācija, t.sk. standartu izstrādāšana lokālajiem tīkliem un pilsēttīkliem. Sk. arī *IEEE 802 Standards*.

829 **instruction**
**instrukcija**
**инструкция**
Semantiska izteiksme programmēšanas valodā, kas apraksta izpildāmo operāciju un identificē tās operandus. Sk. arī *machine instruction*.

830 **integrated circuit (IC)**
**integrētā shēma**
**интегральная схема**
Elektroniska shēma, kas izpilda noteiktas funkcijas un kas izveidota vienā nelielā pusvadītāja kristālā. Izmantojot speciālu tehnoloģiju (kodināšanu, īpašu piedevu ievadīšanu utt.), kristālā tiek izveidoti visi nepieciešamie shēmas elementi (diodes, tranzistori, pretestības u.c.) šo funkciju izpildei. Sk. arī *chip*.

831 **Integrated Drive Electronics (IDE)**
**saskarne *IDE***
**интерфейс *IDE***
Cieto disku diskdziņu saskarnes standarts, ko izmanto *IBM PC* un ar tiem saderīgajos personālajos datoros. Saskarnes *IDE* kontrolleris parasti ir integrēts ar diskdzini. Saskarne *IDE* nodrošina samērā augstu veiktspēju ar zemām izmaksām, jo tās realizēšanai nav nepieciešama speciāla kontrollera karte vai paplašināšanas slots.

832 **Integrated Services Digital Network (ISDN)**
**integrēto pakalpojumu cipartīkls, tīkls *ISDN***
**сеть *ISDN***
Starptautiskās telekomunikāciju savie-

nības komitejas *ITU-T* izstrādāts datu pārraides tīkla standarts, kas nosaka prasības, kurām jāatbilst šī tīkla arhitektūrai un saskarnēm, lai nodrošinātu datu, audio- un vdeoinformacijas pārraidi pa vienotu cipartīklu.

833 **integrated software package**
**integrētās programmatūras pakotne**
**интегрированное программное обеспечение**
Programmatūra, kas ir dažādu lietojumprogrammu (piem., datu bāzu pārvaldības sistēmu, tekstu apstrādes, izklājlapu un datu apmaiņas programmu) apvienojums.

834 **integrity**
**integritāte**
**целостность**
1. Sistēmu, programmu un datu aizsardzība pret netīšu vai ļaunprātīgu bojāšanu vai pārveidošanu.
2. Datora atmiņā uzglabāto datu pilnīguma un korektuma saglabāšana pēc to modificēšanas. Sk. arī *data integrity*.

835 **Intel**
Sk. *Intel Corporation*.

836 **Intel 80286, 80286 mikroprocesors *Intel 80286*, *Intel 80286*, *80286* микропроцессор *Intel 80286*, *80286***
Firmas *Intel* mikroprocesors, kas izmanto 16 megabaitu atmiņu un 16 bitu datu kopni. Šī tipa mikroprocesoram ir divi darba režīmi. Reālrežīmā tas strādā kā *Intel 8086*, izmantojot 1 megabaita atmiņu, bet aizsargrežīmā tiek nodrošināta tieša pieeja 16 megabaitu atmiņai. Uz šī mikroprocesora bāzes firma *IBM* 1984. g. izstrādāja personālo datoru *PC/AT*.

837 **Intel 80386,80386 mikroprocesors *Intel 80386*, *Intel 80386*, *80386* микропроцессор *Intel 80386***
Firmas Intel mikroprocesors, kas izmanto 32 bitu datu kopni un nodrošina tiešu 4 gigabaitu pamatatmiņas adresēšanu. Pazīstamas trīs šī mikroprocesora modifikācijas. *Intel 80386DX* ir 32 bitu reģistri un tas var izmantot 32 bitus atmiņas adresēšanai. Tāpat kā *Intel 80286* arī šis mikroprocesors var strādāt vairākos režīmos. Reālrežīmā tas var izmantot 1 megabaita atmiņu, bet aizsargrežīmā tiek nodrošināta tieša pieeja 4 gigabaitu atmiņai. Atšķirībā no *Intel 80286* aplūkojamais mikroprocesors var strādāt arī virtuālajā režīmā, kas nodrošina tā šķietamu sadalījumu vairākos 8086 tipa mikroprocesoros, katrs no kuriem izmanto 1 megabaita atmiņu un var strādāt ar savu programmu. Uz mikroprocesora *Intel 80386DX* bāzes izveidoti mikroprocesori *Intel 80386SL* un *Intel 80386SX*. Pirmajam no tiem ir pazemināts enerģijas patēriņš un tas ir piemērots izmantošanai portatīvajos datoros, bet otrs, izmantojot 16 bitu iekšējo datu kopni, ļauj lietot lētākas ārējās iekārtas, kas izstrādātas mikroprocesoram *Intel 80286*.

838 **Intel 80486,80486 mikroprocesors *Intel 80486*, *Intel 80486*, *80486* микропроцессор *Intel 80486***
Firmas *Intel* 32 bitu mikroprocesors,

kurš izmanto 64 gigabaitu atmiņu un kurā parasti iebūvēts matemātiskais līdzprocesors un 4 kilobaitu iekšējā kešatmiņa. Izstrādāta virkne šī mikroprocesora modifikāciju (*80486DX*, *80486SL*, *80486SX*), kas, tāpat kā mikroprocesora *Intel 80386* modifikācijas, atšķiras ar ātrdarbību, lietojumu apgabalu un funkcionālajām iespējām.

**839 Intel 8086, 8086 mikroprocesors *Intel 8086*, *Intel 8086, 8086* микропроцессор *Intel 8086***

Firmas *Intel* 16 bitu mikroprocesors, kas izmanto 1 megabaita atmiņu un 16 bitu datu kopni. Mikroprocesors *Intel 8086* izstrādāts 1978. g. un nosaka bāzes arhitektūru visai *Intel 80×86* mikroprocesoru saimei. Sk. arī *Intel Corporation*.

**840 Intel 8088, 8088 mikroprocesors *Intel 8088*, *Intel 8088, 8088* микропроцессор *Intel 8088***

Firmas *Intel* mikroprocesors, kas izmanto 1 megabaita atmiņu, 8 bitu ārējo un 16 bitu iekšējo datu kopni. Šis mikroprocesors tika izmantots, izstrādājot pirmos oriģinālos firmas *IBM* personālos datorus *PC* un *PC/XT*.

**841 Intel Corporation firma *Intel* фирма *Intel***

Firma dibināta 1968. g. ASV un ir viena no pasaules vadošajiem mikroprocesoru un citu pusvadītāju ierīču ražotājiem. Izmantojot mikroprocesoru *Intel 8088*, firma *IBM* jau 1981. g. sāka ražot pirmos personālos datorus. Uz šīs firmas mikroprocesoru *Intel 80286*, *80386*, *80486*, *Intel Pentium* un *Intel Pentium Pro* bāzes ir izstrādāti daudzi firmas *IBM* un citi ar tiem saderīgi personālie datori.

**842 Intel Pentium mikroprocesors *Intel Pentium*, *Intel Pentium*, *Pentium* микропроцессор *Intel Pentium***

Firmas *Intel* 1993. g. izstrādāts 32 bitu superskalārs sarežģītas instrukciju kopas mikroprocesors, kas ir atpakaļsaderīgs ar mikroprocesoru *80×86* saimi. Tam ir 32 bitu adreses un 64 bitu datu kopnes, divi iebūvēti paralēlas darbības skalāri procesori, peldošā komata konveijerprocesors, divas 8 kilobaitu kešatmiņas (viena instrukcijām, bet otra datiem) un zarošanās prognozēšanas iespējas.

**843 Intel Pentium Pro mikroprocesors *Intel Pentium Pro*, *Intel Pentium Pro*, *Pentium Pro* микропроцессор *Pentium Pro***

Firmas *Intel* mikroprocesoru *80×86* saimes superskalārs 32 bitu mikroprocesors, kura izstrāde pabeigta 1995. g. Atšķirībā no mikroprocesora *IntelPentium* tā korpusā kopā ar procesora mikroshēmu iebūvēta arī otrā līmeņa kešatmiņas (256 vai 512 kilobaitu) mikroshēma. Šī tipa mikroprocesoriem ir padziļinātas sazarojumu prognozēšanas iespējas un tos var ērti apvienot vairākprocesoru sistēmās, izmantojot *Pentium Pro* kopni.

**844 interactive processing interaktīvā apstrāde интерактивная обработка**

Tāds datora lietošanas veids, kad visas izpildāmās operācijas parādās displeja

ekrānā. Lietotājs var izlabot kļūdas pirms attiecīgās operācijas izpildes.

**845 interactive video (IV)**
**interaktīvā videosistēma, sistēma *IV***
**интерактивная видеосистема**

Sistēma, ko veido ar datoru savienots videodisku atskaņotājs, kā arī atbilstošā programmatūra. Interaktīvās videosistēmas parasti izmanto mācību procesā.

**846 interface**
**saskarne**
**интерфейс**

Funkciju kopa, kas nodrošina datu apmaiņu starp divām ierīcēm, divām lietojumprogrammām vai starp lietotāju un lietojumprogrammu. Sk. arī *character based user interface, graphical user interface, human-computer interface, interface standard, user interface.*

**847 interface standard**
**saskarnes standarts**
**стандарт интерфейса**

Specifikāciju kopa, kas nosaka divu ierīču savienojumu. Kā šādu ierīču piemērus var minēt diskdziņa kontrolleri un cietā diska vadības elektroniku. Pazīstamākie ir cietā diska saskarņu *ST 506, ESDI* un *SCSI* standarti. Tādi standarti kā firmas *Centronics* saskarne nodrošina savienojumus ar seriālajām un paralēlajām pieslēgvietām.

**848 interlacing**
**rindpārlēces izvērse**
**черезстрочная развертка**

Attēla izspīdināšanas tehnoloģija, kas, izmantojot elektronu staru, pirmajā caurgājienā monitora ekrānā zīmē katru otro rindu, bet otrajā caurgājienā — pārējās. Šī tehnoloģija nodrošina augstāku izšķirtspēju un pazemina displeja izmaksu, bet izsauc vieglu monitora ekrāna ņirboņu. Labākajos datoru displejos parasti izmanto rindsecīgo izvērsi.

**849 interleaved memory**
**mijatmiņa**
**память с расслоением**

Pamatatmiņa, kas ir sadalīta divās vai vairākās sekcijās, pie kurām centrālais procesors var neatkarīgi piekļūt, t.i., var notikt vēršanās pie nākošās sekcijas, kamēr vēl nav pabeigta nolasīšana vai ierakstīšana iepriekšējā. Sk. *memory bank.*

**850 internal font**
**iekšējais fonts**
**внутренний шрифт**

Noteikta burtveidola rakstzīmju kopa, kas iebūvēta printera lasāmatmiņā. Sk. arī *downloadable font, font cartridge, soft font.*

**851 International Business Machines Corporation (IBM)**
**firma *IBM***
**фирма *IBM***

Viena no lielākajām firmām ASV un pasaulē, kas izstrādā un ražo kā datorus, tā programmatūru. Pirmo personālo datoru PC uz mikroprocesora *Intel 8088* bāzes firma *IBM* sāka ražot 1981. g. Vēlāk personālajos datoros *PC XT, PC AT, PS/1* un *PS/2*, kā arī piezīmjdatoros *IBM ThinkPad* izmantoti mikroprocesori *Intel 80286, 80386, 80486* un *Pentium*. Jaunākie personālie datori (*PC 330, PC 830, APTIVA*), darbstacijas (*RS/6000*) un serveri (*A/400*) izveidoti, izmantojot *POWER PC* arhitektūras mikroprocesorus (piem., *MC 601, 604* u.c.). Firma

*IBM* ir pazīstama arī kā arhitektūras *MCA* kopņu izstrādātāja. Lai atvieglotu firmas *IBM* datoru izmantošanu dažādos lietojumos, izstrādāta arhitektūra *SAA*.

**852 International Organization for Standardization (ISO)**
**Starptautiskā standartizācijas organizācija**
**Международная Организация по Стандартизации (МОС)**

Starptautiska organizācija, kas dibināta 1946. g. un kas nodarbojas ar sakaru un informācijas apmaiņas starptautisku standartu izstrādāšanu. Starptautiskā standartizācijas organizācija kopā ar Starptautisko elektrotehnikas komisiju (*IEC*) ir izveidojusi Apvienoto tehnisko komiteju (*JTC-1*), kuras uzdevums ir izstrādāt standartus informācijas tehnoloģijas jomā. Sk. arī *International Telecommunication Union, ITU-T.*

**853 International Telecommunications Union (ITU)**
**Starptautiskā telekomunikāciju apvienība *ITU*, apvienība *ITU***
**Международный союз по электросвязи**

Speciāla Apvienoto Nāciju Organizācijas institūcija, kuras uzdevums ir harmonizēt dažādu valstu nacionālās intereses un veicināt to sadarbību telekomunikāciju jomā.

**854 Internet**
**tīkls *Internet*, *Internet***
**сеть *Internet***

Pasaules lielākais intertīkls, kas sākotnēji izveidojies uz pētnieciskā tīkla *ARPANET* bāzes un sevī apvieno dažādus individuālos datoru tīklus. Savu pašreizējo nosaukumu tīkls ieguvis 80. gadu sākumā. Atsevišķu tīklu mijiedarbība tiek īstenota, izmantojot protokolu *TCP/IP* sistēmu. Tīkls *Internet* strauji paplašinās, aptverot arvien jaunas teritorijas.

**855 internet**
**intertīkls**
**интерсеть**

Saīsināts apzīmējums vairāku datoru tīklu apvienojumam, ko veido, izmantojot maršrutētājus vai citus starptīklu pārveidotājus, un kurā parasti izmanto protokolu *TCP/IP* komplektu. Lai to atšķirtu no globālā tīkla *Internet*, šādu tīklu apvienojuma saīsināto apzīmējumu angļu valodā raksta ar mazo burtu.

**856 Internet account**
**tīkla *Internet* konts**
**учет в сети *Internet***

Uzskaite, ko veic tīkla *Internet* pakalpojumu sniedzējs, ar kura starpniecību lietotājam tiek atļauta pieeja tīkla *Internet* resursiem.

**857 Internet Packet Exchange (IPX)**
**intertīkla pakešapmaiņas protokols, protokols *IPX*, *IPX***
**протокол *IPX***

Firmas *Novell* izstrādātās protokolu sistēmas sastāvdaļa, kas atbilst Atvērto sistēmu sadarbības bāzes etalonmodeļa tīkla slānim. Protokolu *IPX* parasti izmanto datu apmaiņas procesos starp serveriem un darba stacijām. Sk. arī *Sequenced Packet eXchange.*

**858 Internet Protocol (IP)**
**intertīkla protokols *IP*, protokols *IP***
**протокол *IP***

Protokolu *TCP/IP* sistēmas sastāvdaļa, kas atbilst Atvērto sistēmu sadarbības bāzes etalonmodeļa tīkla slānim. Protokols *IP* nodrošina bezsavienojuma režīmu. Saskaņā ar šo protokolu tiek radīta pakešu maršrutēšana to pārsūtīšanā starp dažādiem savstarpēji savienotiem tīkliem.

**859 Internet Service Provider (IPS)**
**tīkla *Internet* pakalpojumu sniedzējs**
**поставщик сервиса в *Internet***
Juridiska persona, kas nodrošina tiešu lietotāja pieeju tīklam *Internet*.

**860 interoperability**
**sadarbspēja**
**способность к взаимодействию**
Divu dažādu ražotāju izgatavotu ierīču vai datoru spēja datoru tīklā apmainīties ar informāciju.

**861 interprocess communication (IPC)**
**starpprocesu komunikācija**
**взаимодействие процессов**
Vispārējs termins datu apmaiņas metožu apzīmēšanai vairākuzdevumu sistēmās, kas realizētas vienā datorā vai tādās programmu sistēmās, kas sadarbojas ar datoru tīkla starpniecību. Sk. arī *Dynamic Data Exchange, Object Linking and Embedding.*

**862 interrecord gap**
**starpierakstu atstarpe**
**зонный интервал**
Tukša telpa starp datu blokiem uz magnētiskā diska vai lentes. Tā parasti veidojas, iedarbinot vai apstādinot disku vai lenti.

**863 interrupt**
**pārtraukums; pārtraukšana**
**прерывание**
Kāda procesa, piem., datora programmas izpildes, apturēšana, kuru izraisa, piem., signāls, ko ģenerē ievadizvades iekārta vai komanda programmā, un kas tiek veikta tā, lai procesa izpildi varētu atsākt. Sk. arī *hardware interrupt, interrupt handler, interrupt vector, software interrupt.*

**864 interrupt handler**
**pārtraukumu apdarinātājs**
**обработчик прерываний**
Speciāla programma (rutīna), kas tiek izpildīta konkrēta pārtraukuma gadījumā. Datora apakšējā atmiņā tiek izveidota tabula ar tās rutīnas adresi, kura veic šī pārtraukuma apdari. Šīs adreses dažkārt sauc par pārtraukuma vektoriem.

**865 interrupt vector**
**pārtraukuma vektors**
**вектор прерывания**
Norāde uz vietu atmiņā, kur tiek glabāta rutīna, kas veic pārtraukumu apkalpošanu. Pārtraukumu vektors satur rutīnas adresi un izmanto to, lai izsauktu rutīnu, kad programmai nepieciešama tās izpilde.

**866 inverse video**
**inversais video**
**обратный видеорежим**
Sk. *reverse video.*

**867 IO.SYS**
**datne *IO.SYS***
**файл *IO.SYS***
Viena no divām apslēptajām sistēmas datnēm, kas instalēta operētājsistēmas *MS-DOS* starta diskā un kurās ir ierīču

draiveri ārējām iekārtām, piem., displejam, tastatūrai, lokanā diska dzinim, cietā diska dzinim, seriālajai pieslēgvietai un reālā laika pulkstenim. Firmas *IBM* versijā šīs datnes nosaukums ir *IBMBIO.COM*

**868 IP**
Sk. *Internet Protocol.*

**869 IP address**
**IP adrese**
**адрес сетевого протокола *IP***
Binārs trīsdesmit divu bitu skaitlis, kas viennozīmīgi identificē konkrēta datora vietu tīklā *Internet*. Katram datoram, kas ir tieši pieslēgts tīklam *Internet*, ir sava *IP* adrese.

**870 IPC**
Sk. *interprocess communication.*

**871 IPX**
Sk. *Internet Packet Exchange.*

**872 IRQ (Interrupt ReQuest)**
**aparatūrpārtraukuma pieprasījums, pieprasījums *IRQ***
**запрос прерывания**
Aparatūrpārtraukums, ko izsauc signāls, kuru nosūta centrālajam procesoram, lai īslaicīgi pārtrauktu normālo datu apstrādes procesu un nodotu vadību pārtraukumu apdarināšanas programmai.

**873 ISA**
Sk. *Industry Standard Architecture.*

**874 ISDN**
Sk. *Integrated Services Digital Network.*

**875 ISO**
Sk. *International Organization for Standardization.*

**876 italic**
**slīpraksts, kursīvs**
**курсив**
Drukas stils, ko parasti izmanto, lai izceltu kādu vārdu, frāzi vai teksta daļu uz pārējā teksta fona. Tajā rakstzīmes ir ieslīpinātas pa labi.

**877 ITU**
Sk. *International Telecommunications Union.*

**878 ITU-T**
**komiteja *ITU-T***
**комитет *ITU-T***
Starptautiskās telekomunikāciju apvienības komiteja, kas nodarbojas ar elektrosakaru standartu izstrādāšanu. Līdz 90. gadu sākumam šī *ITU* apakšvienība bija pazīstama kā Starptautiskā telegrāfijas un telefonijas konsultatīvā komiteja (*CCITT*).

**879 IV**
Sk. *Interactive Video.*

## J

**880 jitter**
**trīce**
**флуктуация**
Attēla neliela vibrācija vai pārvietošanās displeja ekrānā, ko izraisa pārraidītā signāla nestabilitāte.

**881 joystick**
**kursorsvira**
**рычажный указатель**

Kursora pozicionēšanas ierīce, kas dod iespēju lietotājam pārvietot kursoru displeja ekrānā, kustinot vertikālu stienīti. Šī stienīša novirzīšanās jebkurā virzienā no vertikālā stāvokļa izsauc atbilstošu kursora pārvietošanos displeja ekrānā. Kursorsviru visbiežāk izmanto datorspēlēs un atsevišķās datorizētās projektēšanas sistēmās.

882 **jumper**
**tiltslēgs**
**перемычка**
Speciāls slēgs, ko parasti veido plastmasas ligzdā ievietots metāla tiltiņš, ar kura starpniecību tiek savienoti divi blakusstāvošie drukātās plates izvadi, tādējādi mainot tās konfigurāciju un funkcijas.

883 **justification**
**taisnošana**
**выравнивание**
Teksta horizontāla vai vertikāla izlīdzināšana, kurā panāk, ka katras teksta rindiņas pirmā un pēdējā rakstzīme atrodas tai atbilstošajā vietā lappuses malā. Sk. arī *alignment*.

## K

884 **Kb**
Sk. *kilobits*.

885 **KB**
Sk. *kilobyte*.

886 **Kbps**
Sk. *kilobits per second*.

887 **Kermit**
**protokols *Kermit***
**протокол *Kermit***
Protokols, ko izmanto datņu pārsūtīšanai starp personālajiem datoriem un lieldatoriem ar parastā telefonu tīkla starpniecību. Protokols paredz datu plūsmas sadalīšanu blokos, kļūdu atklāšanu un, vajadzības gadījumā, atkārtotu pārraidi.

888 **kerning**
**rakstsavirze**
**установка межзнакового интервала**
Atstarpes samazināšana noteiktu burtu pāru (piem., AT, AV, AW, AY un OY) attēlojumā, lai izdruka būtu labi lasāma un glīti noformēta. Šādos gadījumos ir pieļaujama kādas rakstzīmes daļas attēlošana citai rakstzīmei paredzētajā telpā.

889 **key**
**atslēga; taustiņš**
**ключ; клавиша**
1. Rakstzīmju kopums, kuru izmanto ieraksta identificēšanai datnē, kā arī ātrai pieejai šim ierakstam.
2. Simbolu secība, kuru izmanto šifrēšanas un atšifrēšanas procedūrās, attiecīgi šifrētā un atklātā teksta iegūšanai.
3. Tastatūras konstruktīvs elements, kuru nospiežot, tiek ģenerēts atbilstošās rakstzīmes kods vai iniciēta attiecīgās datora ierīces darbība. Sk. arī *function key*.

890 **key combination**
Sk. *shortcut key*.

891 **key field**
**atslēgas lauks**
**поле ключа**
Viens vai vairāki ieraksta lauki vai atribūts relāciju datu bāzes tabulā, kas

veido atslēgu, kura atvieglo datu izguvi vai atjaunināšanu.

**892 key redefinition**
**taustiņu pārdefinēšana**
**переопределение клавишей**
Citu funkciju piešķiršana atsevišķiem tastatūras taustiņiem. Taustiņu pārdefinēšanu veic īpašas programmas.

**893 keyboard**
**tastatūra**
**клавиатура**
Datora ievadierīce, ko izmanto, lai ar noteiktā veidā izvietotu taustiņu palīdzību ievadītu datorā programmas un datus, kā arī veiktu noteiktas datora vadības funkcijas. Sk. arī *AT keyboard, XT keyboard, QWERTY.*

**894 keyboard buffer**
**tastatūras buferis**
**буфер клавиатуры**
Rezervēts atmiņas apgabals, kurā informācija par taustiņu piesitieniem saglabājas līdz tam laikam, kad programma var tos apstrādāt. Tastatūras buferis nodrošina iespēju strādāt ar tastatūru tajā laikā, kad dators izpilda citus uzdevumus.

**895 keyboard controller**
**tastatūras kontrolleris**
**контроллер клавиатуры**
Tastatūrā iebūvēta viena vai vairākas mikroshēmas, kas uztver taustiņa nospiešanu un ģenerē tai atbilstošu kodu. Tastatūras kontrollera izmantošana palielina datora darbības efektivitāti, atbrīvojot no šī uzdevuma tā centrālo procesoru.

**896 keyboard enhancer**
**tastatūras paplašinātājs**
**расширитель клавиатуры**
Programma, kas pārrauga tastatūras taustiņu nospiešanu un dod iespēju taustiņam (vienam vai vairākiem) piekārtot noteiktu makrodefinīciju.

**897 keyboard processor**
**tastatūras procesors**
**процессор клавиатуры**
Sk. *keyboard controller.*

**898 keyboard template**
**tastatūras veidne**
**трафарет клавиатуры**
Uz datora tastatūras (parasti uz funkcionālajiem taustiņiem) novietots plastikāta vai kāda cita materiāla trafarets, uz kura norādīta informācija par šo taustiņu izmantošanu konkrētam lietojumam.

**899 keypad**
**papildtastatūra**
**вспомогательная клавиатура**
Neliela noteiktas nozīmes taustiņu grupa (piem., ciparu taustiņi), kas parasti novietota personālā datora tastatūras labajā pusē. Sk. arī *numeric keypad.*

**900 keystroke**
**taustiņsitiens**
**нажатие клавиши**
Datora tastatūras taustiņa nospiešana un atlaišana, lai iniciētu kādu darbību vai ievadītu rakstzīmi.

**901 kilo- (K, k)**
**kilo- (K, k)**
**кило-**
1. Prefikss, kas norāda uz kāda lieluma skaitliskās vērtības reizinājumu ar 1000.
2. Prefikss, kas datortehnikā norāda uz reizinājumu ar $2^{10}$ (1024).
*Piezīme.* Lai atšķirtu šīs divas prefiksa "kilo-" nozīmes, pirmajā gadījumā saīsi-

nājumam bieži izmanto simbolu "k", bet otrajā — "K". Šis princips ir izmantots arī šajā vārdnīcā.

902 **kilobit (Kb, Kbit)**
**kilobits (Kb)**
**килобит**
Viens kilobits ir $2^{10}$ jeb 1024 biti. Sk. arī *kilo-*.

903 **kilobits per second (Kbps)**
**kilobiti sekundē**
**килобиты в секунду**
Datu pārraides ātrums, ko mēra ar pārraidīto kilobitu skaitu sekundē.

904 **kilobyte (KB, Kbyte)**
**kilobaits (KB)**
**килобайт**
Viens kilobaits ir $2^{10}$ jeb 1024 baiti. Sk. arī *byte, kilo-*.

## L

905 **LAN**
Sk. *local area network*.

906 **landscape orientation**
**ainavorientācija**
**пейзажноориентированность**
Izdrukājamais dokuments ir ainavorientēts, ja tā platums ir lielāks par augstumu.

907 **LANtastic**
**operētājsistēma *LANtastic*, *LANtastic***
**операционная система *LANtastic***
Lokālo tīklu operētājsistēma, kas domāta tīkliem ar vienādranga arhitektūru. Šādos tīklos nav atsevišķa servera, bet katra stacija var piešķirt savus resursus visām pārējām stacijām, izpildot kā klienta, tā arī servera lomu. Operētājsistēmu *LANtastic* izstrādājusi firma *Artisoft Inc.* un tā darbojas operētājsistēmu *DOS* un *Microsoft Windows*, kā arī *Netware* vidē.

908 **laptop computer**
**klēpjdators**
**портативный (переносной) компьютер**
Viegls, pārnēsājams personālais dators ar atvāžamu plakanu displeju, kas parasti izmanto autonomu barošanas avotu. Tas ir piemērots darbam nestacionāros apstākļos.

909 **laser printer**
**lāzerprinteris**
**лазерное печатающее устройство**
Printeris, kas informāciju uz papīra atveido ar lāzera stara palīdzību, izmantojot elektrostatiskās reprodukcijas tehnoloģiju. Lāzerprinteri nodrošina lielu drukāšanas ātrumu un augstu dokumentu noformēšanas kvalitāti. Sk. arī *electrostatic printer*.

910 **latency**
**latentums**
**время ожидания**
1. Laika intervāls datorā starp datu pieprasījuma brīdi un brīdi, kad tiek uzsākta datu pārsūtīšana.
2. Laika intervāls kādā diska sektors, kurā ir nepieciešamā informācija, sasniedz lasīšanas/rakstīšanas galviņu.
3. Laika intervāls datoru tīklā starp brīdi, kad stacija pieprasa pieeju datu pārraides kanālam, un brīdi, kad kanāls tiek nodots šīs stacijas rīcībā datu pārraidīšanai.

**911 layout**
**izkārtojums**
**размещение**

1. Vispārējs projekta plānojums, kurā var ietilpt sistēmas blokshēma, diagramas, izdruku formas un nepieciešamā dokumentācija.
2. Datu apstrādes sistēmās — rakstzīmju izvietojums ierakstā, datu blokā, datnē vai ziņojumā, kā arī to sakārtojums cietajā kopijā vai displeja ekrānā.
3. Datorizdevniecībā un teksta apstrādē — teksta un grafikas izvietojums lappusē.
4. Datu bāžu sistēmās — norāžu elementu, piem., galveņu vai lauku, izvietojums lappusē.

**912 LCD**

Sk. *liquid crystal display*.

**913 leased line**
**nomātā līnija**
**арендуемая линия**

Sakaru līnija, kuru lietotājs nomā. Nomātā līnija ir gatava tūlītējai lietošanai bez iepriekšējas komutācijas. Tā var nodrošināt kā divpunktu, tā arī vairākpunktu savienojumu un tajā izmanto kopēju datu nesēju.

**914 LED printer**

Sk. *light-emitting diode printer*.

**915 left justify**
**kreisā līdzināšana**
**выравнивать влево**

Sk. *flush left*.

**916 LF**

Sk. *line feed*.

**917 light bar**
**gaismas josla**
**световая полоса**

Izgaismots displeja ekrāna apgabals, kurā atrodas izvēlnē atlasītais elements. Gaismas joslu var veidot ar kādas krāsas palīdzību vai pārvēršot tekstu "melns uz balta" par tekstu "balts uz melna". Sk. arī *highlighting*.

**918 light pen**
**gaismas zīmulis**
**световое перо**

Ievadierīce, kas izmanto gaismjutīgu irbuli, lai zīmētu attēlus displeja ekrānā vai uz grafikas planšetes.

**919 light-emitting diode display**
**gaismas diožu displejs**
**светодиодный дисплей**

Displejs, kas veido rakstzīmes, izmantojot gaismas diožu matricu. Šī tipa displejus pakāpeniski aizstāj ar šķidro kristālu displejiem, kuri patērē mazāk elektroenerģijas.

**920 light-emitting diode printer (LED printer)**
**gaismas diožu printeris, *LED* printeris**
**светодиодный принтер**

Lappušu printeris, kas darbojas līdzīgi lāzerprinterim, bet kurā kā gaismas avotu lāzera vietā izmanto gaismu emitējošas diodes.

**921 LIM EMS**

Sk. *Expanded Memory Specification*.

**922 line**
**rindiņa**
**строка**

1. Tekstu apstrādē — horizontāls lineārs rakstzīmju izvietojums displeja ekrānā vai izdrukā.
2. Programmēšanā — priekšraksts (instrukcija), kas aizņem vienu līniju programmas tekstā.

**923 line adapter**
**līnijas adapteris**
**линейный адаптер**

Adapteris, kas nodrošina datora vai atsevišķa termināļa savienošanu ar sakaru līniju, pārveidojot ciparsignālus formā, kura piemērota to pārsūtīšanai pa sakaru līniju, un pārveidojot pa sakaru līniju saņemtos signālus formā, kas piemērota to apstrādei datorā. Sk. arī *modem*.

**924 line driver**
**līnijas draiveris**
**линейный драйвер**

Iekārta pārraidāmā signāla pastiprināšanai, kas nodrošina signāla pārraidīšanas attāluma palielināšanu. Šo iekārtu izmanto ciparsignālu pārraidei. Tādas iekārtas parasti atrodas abos līnijas galos.

**925 line editor**
**rindiņu redaktors**
**строчный редактор**

Vienkārša tekstu rediģēšanas programma, kas veido un numurē teksta rindiņas un rindsecīgi rediģē tekstu.

**926 line feed**
**rindpadeve**
**перевод строки**

Vadības rakstzīme, kas ziņo printerim, ka tam jāpārvietojas uz nākamo rindiņu, neizmainot iepriekšējo kursora vai drukājošās galviņas pozīciju. Lai kursoru vai printeri nobīdītu uz nākamās rindiņas sākumu, rindpadevei jābūt apvienotai ar rakstatgriezi.

**927 line printer**
**rindprinteris**
**построчно печатающее устройство**

Ātrdarbīgs printeris, kas visu teksta rindiņu drukā vienā darba cikla laikā.

**928 line spacing**
**rindstarpa**
**межстрочный интервал**

Atstatums starp vienas teksta rindkopas rindiņām.

**929 linked object**
**saistītais objekts**
**связный объект**

Dokuments vai tā daļa, kas izstrādāta vienam lietojumam un kas ar sasaistes un iegultes tehnikas palīdzību tiek izmantota citam lietojumam izstrādātā dokumentā. Ja saistītajā dokumentā kaut kas tiek mainīts, bet izstrādājamā dokumenta izveide nav pabeigta, tad standarta *OLE* izmantošana nodrošina šo maiņu automātisku pārnesi izstrādājamajā dokumentā.

**930 liquid crystal display**
**šķidro kristālu displejs**
**дисплей на жидких кристаллах**

Displejs, kas veido rakstzīmes, izmantojot speciālus šķidrumā veidotus šablonus, kuri kļūst necaurspīdīgi, ja tiem pievada spriegumu. Parasti šo tehnoloģiju izmanto ciparpulksteņos, mērinstrumentos un klēpjdatoros, jo tā samazina elektroenerģijas patēriņu.

931 **liquid crystal printer**
**šķidro kristālu printeris**
**принтер на жидких кристаллах**

Elektrostatiskais printeris, ar šķidrā kristāla paneli, caur kuru plūst gaisma. Atkarībā no elektriskās ierosmes pikseļi veido rakstzīmes un caurplūstošā gaisma uz drukājamās virsmas rada elektrostatisku attēlu, kuru pēc tam nostiprina ar pretēji lādētām krāsvielas daļiņām.

932 **list**
**saraksts**
**список**

1. Datu struktūra, kas attēlo sakārtotu ierakstu — saraksta elementu — secību. 2. Datu bāzēs — datu struktūra, kas katram bāzes elementam piekārto rādītāju, kurš norāda attiecīgā datu elementa atrašanās vietu bāzē. Izmantojot sarakstu, lietotājs var dažādos veidos organizēt datus, nemainot to fizisko atrašanās vietu.

933 **list box**
**sarakstlodziņš**
**окно списка**

Lodziņš ar ritināmām izvēlēm, no kurām lietotājs var atlasīt vienu. Sk. arī *combination box*, *drop-down combination box*.

934 **list server**
**sarakstserveris**
**списковой сервер**

Tīkla *Internet* elektroniskā pasta sistēma, kas nodrošina, ka elektroniskā pasta ziņojumi, ko paredzēts nosūtīt kādai adresātu grupai, tiek automātiski nogādāti katram šīs grupas dalībniekam, kurš minēts adresātu sarakstā. Sk. arī *newsgroup*, *USENET*.

935 **loader**
**ielādētājs**
**загрузчик**

Sistēmprogramma, kas izpilda programmas moduļa ielādi no diska brīvpiekļuves atmiņā, kur dinamiski izveido ielādes moduli.

936 **local area network (LAN)**
**lokālais tīkls**
**локальная вычислительная сеть (ЛВС)**

Datoru tīkls, kas izvietots nelielā teritorijā un atrodas lietotāja pārziņā. Lokālais tīkls sastāv no sakaru līnijām, kas savieno personālos datorus un citas elektroniskās koplietošanas iekārtas (printerus, ploterus, datu uzkrāšanas un glabāšanas ierīces). Sk. arī *Ethernet*, *Fiber Distributed Data Interface*, *Token Ring network*.

937 **local bus**
**lokālā kopne**
**локальная шина**

Ātrdarbīga kopne, ko parasti personālajos datoros izmanto datu pārsūtīšanai starp centrālo procesoru un videoadapteriem (piem., kopne *PC* vai kopne *VL*). Lokālo kopni ar zemākas ātrdarbības kopnēm (piem., arhitektūras *ISA*, *EISA*, *MCA* kopnēm) savieno speciāli kopnes tilti.

938 **locked file**
**slēgtā datne**
**блокированный файл**

Datne, kuru nevar modificēt, t.i., nevar pievienot jaunus ierakstus vai izmest vecos.

939 **log**
**žurnāls**
**журнал регистрации**
Datne, kuru operētājsistēma izmanto statistiskās informācijas, dažādu ziņojumu un citu datu vākšanai un uzskaitei.

940 **logical drive**
**loģiskā diskierīce**
**логический дисковод**
Cietā diska sekcija, kas apzīmēta ar kādu burtu un ko lietotājs var izmantot kā atsevišķu disku. Sk. arī *partition*, *physical drive*.

941 **login**
**pieteikšanās**
**вход в систему**
Lietotāja identificēšanās procedūra, pievienojoties datoram pa sakaru līniju. Šīs procedūras laikā dators parasti pieprasa lietotāja vārdu un paroli.

942 **login name**
**pieteikumvārds**
**регистрационное имя**
Unikāls vārds, ko lietotājam piešķir sistēmas (datora tīkla) administrators un ko izmanto, lai identificētu lietotāju. Šis vārds kopā ar paroli dod lietotājam iespēju piekļūt sistēmai.

943 **login script**
**pieteikumskripts**
**сценарий регистрации**
Instrukciju saraksts, ar kura starpniecību zvanītājprogramma īsteno iezvanpieeju: uzgriež pakalpojumu piegādātāja numuru, nodod lietotāja pieteikumvārdu un paroli, kā arī nodibina nepieciešamo savienojumu.

944 **Logo**
**valoda *Logo***
**язык *Logo***
Augsta līmeņa programmēšanas valoda, kas viegli lietojama un īpaši piemērota programmēšanas mācīšanai bērniem. Viens no šīs valodas galvenajiem lietojumiem ir bruņurupučgrafika. Grafisko attēlu veidošanu panāk ar programmas instrukcijām, kas liek bruņurupucim sagatavoties zīmēšanai, pārvietoties augšup, lejup, uz priekšu, atpakaļ, pa labi, pa kreisi utt. Apgūstot programmēšanas iemaņas valodā *Logo*, tiek sagatavota pāreja uz sarežģītākas programmēšanas valodas (piem., *LISP*) izmantošanu.

945 **logoff**
**atteikšanās**
**выход из системы**
Datorsakaru seansa pārtraukšanas procedūra. Sk. arī *login*.

946 **logon**
Sk. *login*.

947 **logon file**
**pieteikumdatne**
**файл регистрации**
Pakešdatne vai konfigurācijas datne, kas iniciē lokālā tīkla programmatūras darbību un nodibina savienojumu ar tīklu pēc darba stacijas ieslēgšanas.

948 **logout**
Sk. *logoff*.

949 **look-up table**
**pārlūktabula**
**справочная таблица**
Datora atmiņā uzglabājamo datu izkār-

tojums, kas izklājprogrammai ļauj ātri piekļūt vajadzīgajiem, iepriekš sagatavotajiem datiem.

**950 lossless compression**
**bezzudumu saspiešana**
**сжатие без потерь**

Attēlu saspiešanas metode, ar kuru samazina bitu skaitu, ko izmanto attēla katra pikseļa veidošanai. Saspiešanas rezultātā netiek zaudēta ne informācija, ne attēla asums.

**951 lossy compression**
**zudumradošā saspiešana**
**сжатие с потерями**

Attēlu saspiešanas metode, kura ļauj attēlā samazināt katra pikseļa veidošanai izmantojamo bitu skaitu, bet kuras izmantošana ir saistīta ar informācijas zudumiem.

**952 lost cluster**
**zaudēts klasteris**
**потерянный кластер**

Diska sektoru grupa, ko operētājsistēma uzrāda kā aizņemtus, bet kam neviena datnes daļa nav piekārtota. Zaudēti klasteri rodas gadījumā, ja, veidojot datni, notiek operētājsistēmas darbības pārtraukums.

**953 Lotus 1-2-3**
**izklājprogramma *Lotus 1-2-3*, *Lotus 1-2-3***
**программа *Lotus 1-2-3***

Plaši pazīstama izklājprogramma, ko izstrādājusi firma *Lotus Development Corporation*. *Lotus 1-2-3* bija pirmā izklājprogramma, kas bez tradicionālajām izklājlapu funkcijām nodrošināja arī zināmas datņu pārvaldības, grafikas un teksta apstrādes iespējas.

**954 low memory**
**apakšējā atmiņa**
**нижняя память**

Personālajos datoros, kuru pamatatmiņa ir 1 megabaits, par apakšējo atmiņu sauc pamatatmiņas daļu līdz 640 kilobaitiem. Apakšējā atmiņa veido brīvpieejas atmiņu, ko kopīgi izmanto operētājsistēma *MS-DOS* un lietojumprogrammas. Sk. arī *high memory*.

**955 low radiation screen**
**vāja starojuma ekrāns**
**экран с низким излучением**

Monitora ekrāns ar zemu magnētiskā un elektriskā starojuma intensitāti. Parasti šis termins norāda uz atbilstību kādam noteiktam firmas vai valsts standartam.

## M

**956 machine instruction**
**mašīninstrukcija**
**машинная команда**

Instrukcija, ko dators var tieši identificēt un izpildīt. Sk. arī *command*.

**957 machine language**
**mašīnvaloda**
**машинный язык**

Valoda, kuru pazīst un kuras komandas izpilda datora centrālais procesors. Programmām, kas rakstītas mašīnvalodā, nav nepieciešama iepriekšēja kompilēšana.

**958 machine-readable data**
**mašīnlasāmi dati**
**машиночитаемые данные**

Jebkura veida dati, kurus ar ievadierīci var ievadīt datorā. Kā šādu datu piemērus var minēt datus, kas ierakstīti

magnētiskajos diskos un magnētiskajās lentēs, kā arī svītrkodus vai noteiktus burtveidolus, ja datora aprīkojumā ir skeneris un optisko rakstzīmju pazīšanas programmatūra.

959 **Macintosh**
**personālo datoru saime**
***Macintosh*, *Macintosh* saime**
**семейство персональных компьютеров *Macintosh***

Firmas *Apple Computer, Inc.* personālo datoru saime, kas izstrādāta uz mikroprocesora *Motorola 68000*, kā arī arhitektūras *Power PC* bāzes. Šīs saimes personālajiem datoriem piemīt plašas grafikas un tekstu apstrādes iespējas. To veiktspēja ir salīdzināma ar *IBM* personālo datoru veiktspēju, bet tie nav savstarpēji saderīgi.

960 **macro**
**makrodefinīcija, makro**
**макроопределение**

Izvēlņu virkņu, ar taustiņiem iniciējamu darbību un instrukciju kopums, kas pierakstīts un saglabāts, piešķirot tam noteiktu vārdu vai taustiņu kombināciju. Lietojot programmā šo makrodefinīcijas vārdu vai nospiežot attiecīgo taustiņu, tiek izpildītas visas makrodefinīcijā paredzētās darbības. Makrodefinīcijas veido, lai bieži izmantojamu vienveidīgu darbību virkņu ievadīšanu aizstātu ar vienu taustiņsitienu vai lai izveidotu lietojumprogrammā miniatūras apakšprogrammas.

961 **magnetic disk**
**magnētiskais disks**
**магнитный диск**

Sk. *disk*.

962 **magneto-optical disc**
**magnētoptiskais disks**
**магнитнооптический диск**

Datora atmiņas ierīce, kas apvieno optiskā un magnētiskā diska īpašības. Magnētooptiskajā diskā var uzglabāt ievērojamu informācijas daudzumu, bet informācijas nolasīšanas ātrums šī tipa diskiem ir relatīvi mazs.

963 **mail bridge**
**pasta tilts**
**мост электронной почты**

Speciāls pasta vārtejas veids, ko izmanto elektroniskā pasta ziņojumu pārsūtīšanai no viena datoru tīkla otrā. Pārsūtāmajiem ziņojumiem jāpakļaujas noteiktiem administratīviem kritērijiem.

964 **mail gateway**
**pasta vārteja**
**шлюз электронной почты**

Vārteja, kas savieno divas vai vairākas (iespējams, arī atšķirīgas) elektroniskā pasta sistēmas un nodrošina ziņojumu pārsūtīšanu starp tām. Dažkārt pasta vārteja uzkrāj no vienas sistēmas saņemto pilno ziņojumu un tikai tad veic tā translēšanu un tālāko nosūtīšanu adresātam.

965 **mail merge**
**vēstuļu sapludināšana**
**слияние писем**

Process, kurā ar speciālas programmatūras starpniecību noformē elektroniskās vēstules. Izmantojot vēstuļu sapludināšanas sistēmu, lietotājs vispirms kādā datnē uzkrāj adrešu un adresātu sarakstu, kā arī citus bieži izmantojamus datus. Citā datnē tiek veidota pati vēstule, kurā adresi, adresāta vārdu un citu

papildinformāciju aizstāj ar speciāliem kodiem. Sapludināšanas programma šos simbolus automātiski aizstāj ar pirmās datnes fragmentiem.

**966 mail server**
**pasta serveris**
**сервер электронной почты**

Serveris, kas ir atbildīgs par elektroniskā pasta ziņojumu sadali adresātiem. Pasta servera programmas veic lietotāju pierakstīšanu adresātu sarakstā un izrakstīšanu no tā un nodrošina pieprasītās informācijas nosūtīšanu.

**967 mailbox**
**pastkastīte**
**почтовый ящик**

Atmiņas apgabals pa elektronisko pastu saņemto dokumentu vai datu glabāšanai. Parasti katram lietotājam ir sava pastkastīte, kurā automātiski tiek ievietots tam adresētais ziņojums.

**968 mailing list**
**adresātu saraksts**
**список рассылки**

Elektroniskā pasta adrešu saraksts, ko izmanto, lai pārsūtītu ziņojumus cilvēku grupai, kuru vieno kādas noteiktas intereses. Lai pievienotos šai grupai, parasti jāizpilda kāda noteikta procedūra. No adresātu saraksta var izstāties, nosūtot speciālu paziņojumu.

**969 main memory**
**operatīvā atmiņa, pamatatmiņa**
**оперативное запоминающее устройство**

Ar datu apstrādes procesoru tieši saistīta atmiņa, kurā līdz pārsūtīšanai palīgatmiņā glabājas izpildāmā programma, starprezultāti un dati. Operatīvās atmiņas ātrdarbība ir salīdzināma ar centrālā procesora ātrdarbību. Sk. arī *Random Access Memory*.

**970 main storage**

Sk. *main memory*.

**971 maintainability**
**uzturamība**
**сопровождаемость**

1. Programmatūras vai tās komponentu īpašība, kas nodrošina iespēju tos modificēt, lai labotu kļūdas, uzlabotu veiktspēju un citus raksturojumus, kā arī lai adaptētu tos funkcionēšanas videi. Sk. arī *expandability*, *flexibility*.
2. Aparatūras vai tās komponentu īpašība, kas nodrošina tās spēju saglabāt vai atjaunot tādu stāvokli, kurā tā izpilda uzdotās funkcijas.

**972 maintenance**
**uzturēšana**
**сопровождение**

1. Programmu sistēmas vai tās komponentu modificēšana, lai labotu kļūdas, uzlabotu veiktspēju vai adaptētu to mainītai funkcionēšanas videi.
2. Aparatūras vai tās komponentu saglabāšana vai restaurēšana tādā stāvoklī, kurā tā izpilda uzdotās funkcijas.

**973 major key**
**vecākā atslēga**
**главный ключ**

Sk. *primary key*.

**974 male connector**
**vīrišķais savienotājs**
**вилка разъема**

Datora kabeļa pieslēgvietas konstrukcija,

kas ar spraudņu palīdzību ļauj kabeli saslēgt ar sievišķo savienotāju.

**975 MAN**

Sk. *Metropolitan Area Network*.

**976 mapping**
**kartēšana**
**отображение**

1. Vienā formātā kodētu datu pārveidošana citā formātā kodētos datos.
2. Kādas objektu kopas pārsūtīšana no vienas vietas otrā, piem., programmu moduļu pārsūtīšana no diskatmiņas operatīvajā atmiņā vai viena tipa adrešu pārveidošana cita tipa adresēs.

**977 margin**
**mala**
**край**

Teksta apstrādes izdruku baltie laukumi ārpus galvenā teksta masīva lappuses augšā, apakšā un sānos.

**978 mark**
**atzīme**
**метка**

1. Simbols vai simbolu secība, kas nosaka datnes, datu bloka, ieraksta vai vārda sākumu vai beigas.
2. Speciāla rakstzīme, ko izmanto teksta procesors, lai atšķirtu rediģēto tekstu no sākotnējā.
3. Vizuāls palīglīdzeklis, kas uz izvēlnēm balstītās programmās norāda konkrēto izvēlni.
4. Ar zīmuli uzvilkta līnija, ko atšifrē optiskā lasāmierīce.

**979 mask**
**maska**
**маска**

1. Vārds, ko veido bitu, baitu vai simbolu apvienojums un ko lieto salīdzināšanai ar citu tāda paša formāta datu struktūru, lai varētu saglabāt, izmest vai neņemt vērā noteiktus bitus vai sinbolus.
2. Rakstzīmju šablons, kas ierobežo noteiktu rakstzīmju izmantošanu kādā datu laukā.
3. Process, kurā izvēlas atsevišķus apgabalus, kam jāpaliek nemainītiem, ja pats attēls tiek pārveidots.

**980 mass storage**
**lielapjoma atmiņa**
**массовая память**

Palīgatmiņa lielu, relatīvi reti izmantojamu datu masīvu glabāšanai. Sk. arī *disk*, *floppy disk*, *hard disk*.

**981 math coprocessor**
**matemātiskais līdzprocesors**
**математический сопроцессор**

Līdzprocesors, kas specializēts ātriem matemātiskiem aprēķiniem. Matemātiskais līdzprocesors var būt izveidots kā atsevišķa mikroshēma vai arī tas var būt apvienots vienā mikroshēmā ar mikroprocesoru, kas izpilda centrālā procesora funkcijas. Šī līdzprocesora izmantošana ļauj ievērojami paātrināt matemātisko aprēķinu izpildi.

**982 matrix printer**
**matricprinteris**
**матричное печатающее устройство**

Printeris, kas rakstzīmes drukā kā punktveida rastru. Sk. *dot matrix printer*.

**983 maximize**
**maksimizēšana**
**максимизация**

Lietotāja grafiskās saskarnes termins, ar ko apzīmē loga palielināšanu līdz visa displeja ekrāna apmēriem. Sk. arī *minimize*.

**984 Mb**
Sk. *megabit*.

**985 MB**
Sk. *megabyte*.

**986 MCA**
Sk. *Micro Channel Architecture*.

**987 MCGA**
Sk. *Multicolor Graphics Array*.

**988 MCI**
Sk. *Media Control Interface*.

**989 MDA**
Sk. *Monochrom Display Adapter*.

**990 mechanical mouse**
**mehāniskā pele**
**механическая мышь**
Pele, par kuras kustības virzienu informāciju iegūst ar speciālas peles apakšējā virsmā iebūvētas lodītes palīdzību. Peles pārvietošana pa kādu virsmu rada šīs lodītes griešanos. Lodītes griezes kustība tiek pārveidota attiecīgos elektriskajos signālos, ko pievada datoram un izmanto kursora kustības vadībai displeja ekrānā. Sk. arī *joystick, optomechanical mouse, trackball*.

**991 Media Control Interface (MCI)**
**vides vadības saskarne, saskarne *MCI*, *MCI***
**интерфейс управления средой**
Aparatūrneatkarīga programmēšanas saskarne, ko izstrādājušas firmas *Microsoft* un *IBM* un kas nodrošina multivides iekārtu vadības pamatfunkcijas (piem., atskaņošanu, apturēšanu, pārtīšanu).

**992 Media Player**
**utilīta *Media Player***
**утилита *Media Player***
Operētājsistēmas *Microsoft Windows* palīgprogramma, kas, izmantojot ekrānā redzamos taustiņu attēlus, ļauj lietotājam vadīt sistēmā instalēto atskaņotājaparatūru (piem., *CD-ROM* atskaņotāju).

**993 medium**
**vide**
**среда**
Materiāls, uz kura var saglabāt datus vai ar kura starpniecību tos var pārraidīt. Kā šādas vides piemērus var minēt magnētisko disku un lenti, disketi, koaksiālo kabeli, vīto pāri u.c.

**994 mega- (M)**
**mega- (M)**
**мега-**
1. Prefikss, kas metriskajā sistēmā norāda uz kāda lieluma skaitliskās vērtības reizinājumu ar 1 miljonu ($10^6$).
2. Prefikss, kas datortehnikā norāda uz kāda lieluma skaitliskās vērtības reizinājumu ar $2^{20}$ (1 048 576).

**995 megabit (Mb, Mbit)**
**megabits (Mb)**
**мегабит**
Viens megabits ir $2^{20}$ jeb 1 048 576 biti. Sk. arī *mega-*.

**996 megabyte (MB)**
**megabaits (MB)**
**мегабайт**

Viens megabaits ir $2^{20}$ jeb 1 048 576 baiti. Sk. arī *mega-*.

997 **memory**
**atmiņa**
**память**
Iekārta vai datu vide, kurā var uzglabāt informāciju tās vēlākai izmantošanai. Sk. arī *cache memory, conventional memory, expanded memory, extended memory, flash memory, high memory, low memory, main memory, non-volatile memory, Random Access Memory, Read-Only Memory, virtual memory, volatile memory.*

998 **memory bank**
**atmiņas banka**
**банк памяти**
Fizikāla atmiņas sekcija. Sk. *interleaved memory.*

999 **memory chip**
**atmiņas mikroshēma**
**микросхема памяти**
Integrētā shēma, kas īslaicīgi (brīvpieejas atmiņā), pastāvīgi (lasāmatmiņā, programmējamā lasāmatmiņā) vai arī līdz brīdim, kad nepieciešams to mainīt (pārprogrammējamā lasāmatmiņā), saglabā tajā ierakstītās programmas vai datus.

1000 **memory dump**
**atmiņas izmete**
**разгрузка памяти**
Sk. *dump.*

1001 **memory management**
**atmiņas pārvaldība**
**управление памятью**
Atmiņas vadības līdzekļu un metožu kopums, kas ļauj šķietami palielināt brīvpieejas atmiņas apjomu, veidojot izvērsto atmiņu, paplašināto atmiņu vai virtuālo atmiņu, ko var izmantot programmu izpildei.

1002 **Memory Management Unit**
**atmiņas pārvaldības bloks**
**блок управления памятью**
Shēma, kas pārveido virtuālās atmiņas adreses reālās atmiņas adresēs, kā arī veic citas atmiņas administratīvās vadības funkcijas.

1003 **memory map**
**atmiņas kartējums**
**карта распределения памяти**
Diagramma, kas attēlo datu un programmu izvietojumu datora atmiņā.

1004 **menu**
**izvēlne**
**меню**
Dažādu režīmu, programmu, instrukciju vai atbilžu saraksts, kas interaktīvās sistēmās tiek attēlots displeja ekrānā un piedāvāts lietotāja izvēlei. Izvēlētais variants nosaka tālāko sistēmas darbību.

1005 **menu bar**
**izvēlņu josla**
**строка меню**
Taisnstūrveida josla, kas parasti atrodas displeja ekrāna vai loga augšējā daļā un kurā norādīti pieejamo izvēlņu vārdi.

1006 **menu item**
**izvēlnes elements**
**элемент меню**
Atsevišķa komanda vai izvēle, kas parādās izvēlnē.

1007 **menu name**
**izvēlnes vārds**
**имя меню**

Nosaukums, ar kādu izvēlne parādās izvēlņu joslā.

**1008 menu-driven program**
**izvēļņvadāma programma**
**программа, управляемая с помощью меню**

Programma, kas savas darbības gaitā nodrošina lietotāju ar izvēlnēm, no kurām viņš var izvēlēties attiecīgas komandas vai programmopcijas, tādējādi atbrīvojot lietotāju no komandu iegaumēšanas.

**1009 message**
**ziņojums**
**сообщение**

1. Rakstzīmju virkne, kas izveido datu bloku, ko pārraida datoru tīklos. Ziņojums parasti sastāv no ziņojuma iesākuma, rumpja un ziņojuma noslēguma.
2. Speciālā veidā formalizēts dokuments, ko pārraida, izmantojot elektronisko pastu.
3. Displeja ekrānā izspīdināts teksts, kas ziņo lietotājam par kādu nosacījumu vai programmu.
4. Dati, kas, izmantojot saskarni *MIDI*, tiek nosūtīti kāda mūzikas instrumenta ierīces vadībai.

**1010 message box**
**ziņojumlodziņš**
**окно сообщения**

Lodziņš, kas parādās displeja ekrānā kādas programmas izpildes gaitā un parasti informē par kļūdām vai brīdina par lietotāja veikto darbību iespējamām sekām. Sk. arī *dialog box, list box.*

**1011 Metropolitan Area Network (MAN)**
**pilsētttīkls**
**городская сеть**

Datoru tīkls, kas parasti aptver kādas pilsētas vai tās lielākās daļas teritoriju un apvieno tajā izvietotos lokālos tīklus. Pilsēttīklā uzbūves principus nosaka *IEEE 802.6* standarts. Sk. arī *Distributed Queue Dual Bus, Fiber Distributed Data Interface, IEEE 802 Standards.*

**1012 MFM**

Sk. *Modified Frequency Modulation.*

**1013 Micro Channel Architecture (MCA)**
**mikrokanāla arhitektūra, arhitektūra *MCA***
**архитектура *MCA***

Datoru kopnes arhitektūra, ko firma *IBM* 1987.g. izstrādāja un ieviesa personālajos datoros *IBM PS/2*. Tā nodrošina gan asinhronu, gan sinhronu 32 bitu datu pārraidi ar decentralizētu arbitrāžu un var strādāt sprādzienrežīmā. Kopnei var būt vairāki vedēji. Fiziski tā nav savietojama ar kopnēm, kas izveidotas atbilstoši arhitektūras *ISA* un *EISA* prasībām, jo karšu izmēri ir dažādi.

**1014 micro-**
**mikro-**
**микро-**

1. Viena miljonā daļa (piem., mikrosekunde).
2. Prefikss, kas apzīmē kaut ko mazu vai kompaktu (piem., mikroprocesors, mikroshēma).

*Piezīme*: angļu valodā — arī mikroskaitļotāja saīsināts nosaukums.

**1015 Microcom Networking Protocols (MNP)**
**firmas *Microcom* tīklošanas protokoli, protokoli *MNP***
**протоколы *MNP***

Firmas *Microcom* protokolu saime, ko izmanto, pārraidot datus pa sakaru līnijām ar modemu starpniecību. Dažādas šo protokolu versijas praktiski ir kļuvušas par kļūdu atklāšanas, labošanas un dažādas pakāpes datu saspiešanas procedūru standartiem.

1016 **microcomputer**
**mikrodators,**
**mikroskaitļotājs**
**микро ЭВМ, микрокомпьютер**

Dators, kura centrālais procesors realizēts, izmantojot mikroprocesoru. Pie mikrodatoriem pieder, piem., personālie datori. Sk. *laptop computer*, *notebook computer*, *palmtop computer*, *pocket computer*.

1017 **microprocessor**
**mikroprocesors**
**микропроцессор**

Augstas integrācijas pakāpes shēma, kas izpilda centrālā procesora funkcijas. Mikroprocesorus plaši izmanto mikrodatoros, tirdzniecības un sadzīves tehniskajās ierīcēs, rotaļlietās u.c.

1018 **Microsoft Access**
**datu bāzes pārvaldības sistēma *Microsoft Access*,**
**sistēma *Microsoft Access***
**система управления базой данных *Microsoft Access***

Datu bāzu pārvaldības sistēma *Microsoft Windows* vidē. Dati sistēmā *Microsoft Access* tiek saglabāti tabulās. Sistēma nodrošina "vilkt un nomest", kā arī piemērvaicājumu iespēju. Tajā ir augsta līmeņa programmēšanas valoda, kas izmanto kompilatoru *Visual Basic*. Datu bāzu vadības sistēma var darboties vairāklietotāju režīmā un izmantot attālus datus. Visas minētās iespējas sistēma *Microsoft Access* piedāvā realizēt ar iebūvētu vedni.

1019 **Microsoft Corp.**
**firma *Microsoft***
**фирма *Microsoft***

Viena no vadošajām personālo datoru programmatūras izstrādātajām firmām ASV un pasaulē. Šī firma ir pazīstama kā operētājsistēmu *PC-DOS* un *MS-DOS*, kā arī dažādu operētājsistēmas *Microsoft Windows* versiju (*Windows 3.0*, *Windows 3.1*, *Windows NT*, *Windows 95*) izstrādātāja. Firmas *Microsoft* izstrādājumu klāstā ir arī daudz dažādu lietojumprogrammu.

1020 **Microsoft DOS**

Sk. *MS-DOS*.

1021 **Microsoft Excel**
**izklājlapu programma *Microsoft Excel*, *Microsoft Excel***
**программа табличных вычислений *Microsoft Excel***

Firmas *Microsoft* izveidota izklājlapu apstrādes programma, kas lietojama firmu *IBM* un *Apple Macintosh* datoriem. Tā var apvienot vairākas izklājlapas un nodrošina lietotājam plašu grafisko līdzekļu klāstu.

1022 **Microsoft Office**
**biroja programmatūra *Microsoft Office*, *Microsoft Office***
**система автоматизации учрежденческой деятельности *Microsoft Office***

Plaši izmantots firmas *Microsoft* biroja programmu komplekts, kurā ietilpst vārdu procesors *Microsoft Word*, izklājlapu

programma *Microsoft Excel*, elektroniskā pasta un prezentēšanas grafikas programmas. *Microsoft Office* profesionālajā versijā šīm programmām pievienota datu bāzu pārvaldības sistēma *Microsoft Access*.

**1023 Microsoft Windows**
**operētājsistēma *Microsoft Windows, Microsoft Windows***
**операционная система *Microsoft Windows***

Firmas *Microsoft* uz operētājsistēmas *MS-DOS* bāzes 1983. g. izveidota vairākuzdevumu operētājsistēma, kas nodrošina grafisko lietotāja saskarni, t.sk., ikonu, dialoglodziņu, izvelkamo izvēlņu un peles izmantošanu. Operētājsistēmas *Microsoft Windows* galvenie komponenti ir lietojumprogrammas *Print Manager* un *File Manager*, kā arī čaula *Program Manager*. Sk. arī *Microsoft Windows 95*, *Microsoft Word*.

**1024 Microsoft Windows 95**
**operētājsistēma *Microsoft Windows 95, Microsoft Windows 95***
**операционная система *Microsoft Windows 95***

Operētājsistēma, kas paredzēta datoriem, kuri izveidoti, izmantojot mikroprocesorus *Intel 80486* un *Intel Pentium*. Šī operētājsistēma nodrošina darbu visām operētājsistēmas *DOS* un *Microsoft Windows* lietojumprogrammām, ieskaitot multivides lietojumus, kā arī izmanto uzlabotu grafisko lietotāju saskarni, kas atvieglo tās ikdienas lietojumus. Operētājsistēma *Microsoft Windows 95* ir relatīvi vienkārša, ātrdarbīga un jaudīga 32-bitu vairākuzdevumu operētājsistēma ar iebūvētiem līdzekļiem, kas atvieglo jaunu datoru tīklu veidošanu vai pieslēgšanos esošajiem tīkliem.

**1025 Microsoft Word**
**vārdu procesors *Microsoft Word, Microsoft Word***
**текстовый процессор *Microsoft Word***

Teksta apstrādes programmatūra, ko izstrādājusi firma *Microsoft* un kas paredzēta izmantošanai gan firmas *IBM* personālajos datoros, gan arī firmas *Apple Macintosh* saimes datoros. Šīs programmatūras versija nodrošina gan grafisko, gan tekstuālo saskarni darbam ar dokumentiem.

**1026 microspacing**
**mikrostarpināšana**
**микрорасположение**

Mainīga platuma starpu izmantošana rakstzīmju atdalīšanai dokumentā un vārdu izlīdzināšanai tekstā.

**1027 MIDI**

Sk. *Musical Instrument Digital Interface*.

**1028 MIME**

Sk. *Multipurpose Internet Mail Extension*.

**1029 minimize**
**minimizēšana**
**минимизация**

Grafiskajā lietotāja saskarnē — loga samazināšana līdz tā pārveidošanai ikonā. Sk. arī *maximize*.

**1030 minor key**
**jaunākā atslēga**
**младший ключ**

Sk. *alternate key*.

1031 **MNP**

Sk. *Microcom Networking Protocols*.

1032 **modeling**
**modelēšana**
**моделирование**

Procesu, sistēmu vai to darbības aprakstīšana vai attēlošana ar matemātisku modeļu palīdzību. Sk. arī *simulation*.

1033 **modem (modulator — demodulator)**
**modems**
**модем**

Funkcionāla ierīce, kas ciparsignālu pārveido analogsignālā un analogsignālu ciparsignālā, lai tos varētu apstrādāt un pārraidīt pa sakaru līnijām. Modemus lieto attālu termināļu pievienošanai datoru tīklam.

1034 **Modified Frequency Modulation (MFM)**
**modificētā frekvences modulācija, *MFM* tehnoloģija**
**модифицированная частотная модуляция**

Ierakstes tehnoloģija, kas izmanto frekvences modulāciju, pozicionējot katru magnētiskajā diskā vai lentē ierakstāmo bitu atbilstoši iepriekš ierakstītajiem. Tas dod iespēju neizmantot speciālus sinhronizēšanas bitus un, salīdzinot ar vienkāršo frekvences modulāciju, divkāršot pieraksta blīvumu. Sk. *frequency modulation*.

1035 **modularity**
**modularitāte**
**модульность**

Īpašība, kas liecina, ka sistēma vai datoru programma no atsevišķiem funkcionāliem komponentiem (moduļiem) ir izveidota tā, lai izmaiņas kadā no šiem komponentiem praktiski neiespaidotu citu komponentu darbību.

1036 **monitor**
**monitors; pārraugs**
**монитор**

1. Displeja, kas parasti izveidots, izmantojot katodstaru lampu, konstruktīvs bloks, kas ekrānā atveido vienas vai vairāku krāsu attēlus. Monitors ir apgādāts ar visām iekšējām shēmām šo funkciju veikšnai. Praksē bieži terminus "monitors" un "displejs" lieto kā sinonīmus. Sk. arī *analog monitor*, *color monitor*, *digital monitor*, *monochrome monitor*.

2. Programma, kas pārrauga citu programmu darbību datorā.

1037 **Monochrome Display Adapter (MDA)**
**videostandarts *MDA*; videodapteris *MDA*, adapteris *MDA***
**монохромный дисплейный адаптер**

1. Videostandarts, kas izstrādāts firmas *IBM* personālajiem datoriem un paredzēts izmantošanai tikai vienkrāsas teksta režīmā, nodrošinot visai augstu izšķirtspēju 720×350.

2. Videoadapteris, kas atbilst videostandartam *MDA* un, izmantojot piemērotu vienkrāsas monitoru, nodrošina tā prasību izpildi.

1038 **monochrome monitor**
**vienkrāsas monitors**
**монохромный монитор**

Monitors, kas var veidot attēlu displejā, izmantojot tikai vienu krāsu. Šie monitori nodrošina lielāku burtu un ciparu

izšķirtspēju nekā krāsu monitori, un tāpēc tos parasti izmanto tekstu apstrādei. Sk. arī *color monitor*.

### 1039 monospace font
### vienplatuma fonts
### равноразмерный шрифт

Noteikta lieluma un stila rakstzīmju kopa, kas izdrukā neatkarīgi no katras rakstzīmes platuma aizņem vienāda horizontālā izmēra laukumu. Sk. arī *fixed pitch*, *monospacing*, *proportional font*, *proportional pitch*, *proportional spacing*.

### 1040 monospacing
### vienplatumošana
### равноразмерная фиксация ширины

Vienāda platuma telpas iedalīšana rakstzīmju drukāšanai vai attēlošanai ekrānā neatkarīgi no to formas (piem., burti m un i).

### 1041 morphing
### metamorfēšana
### метаморфоза

Process, kas vienu attēlu pārveido citā. Šis process notiek noteiktā laika intervālā un parasti ir saistīts ar dažādu speciālo efektu veidošanu.

### 1042 Mosaic
### pārlūkprogramma *Mosaic*, *Mosaic*
### программа ускоренного просмотра *Mosaic*

Globālā tīmekļa *WWW* pārlūkprogramma, kas izstrādāta Ilinoisas universitātē ASV. Lai nodrošinātu pieeju tīkla *Internet* resursiem, pārlūkprogramma *Mosaic* izmanto grafisko lietotāja saskarni. Ar peles palīdzību tā ātri un ērti ļauj īstenot hiperteksta dokumentu pārlūkošanas procesu. Sk. arī *Netscape Navigator*.

### 1043 mother board
### mātesplate
### объединительная плата

Mikrodatora montāžas plate, kas veido mikrodatora pamatstruktūru. Tajā ir centrālais procesors, operatīvā atmiņa, virknes un paralēlās pieslēgvietas dažādu ārējo iekārtu (piem., displeja, tastatūras un disku) vadībai. Ja mikroshēma, kas vada displeju, virknes un paralēlās pieslēgvietas, peli un diskdziņus, neatrodas uz mātesplates, tad tos vada autonomas vadības iekārtas, kuras pievieno mātesplates izvērses platei.

### 1044 Motorola 68000 mikroprocesoru saime
### *Motorola 68000*, *Motorola 68000*
### семейство микропроцессоров *Motorola 68000*

Firmas *Motorola* 32 bitu mikroprocesoru saime, ko izmanto *Macintosh* saimes personālajos datoros un daudzās darbstacijās. Šīs saimes mikroprocesoru raksturīgākās īpašības ir šādas:

— 68000 var adresēt 16 megabaitu atmiņu, un tam ir 16 bitu datu kopne;

— 68020 var adresēt 4 gigabaitu atmiņu, un tam ir 32 bitu datu kopne;

— 68030 var adresēt 4 gigabaitu atmiņu, un tam arī ir 32 bitu datu kopne, bet tas ir ātrāks par 68020 un tajā ir iebūvēta kešatmiņa;

— 68040 ir līdzīgs 68030, bet aptuveni 3 reizes ātrāks.

1045 **Motorola, Inc.**
**firma *Motorola***
**фирма *Motorola***

Viena no vadošajām pusvadītāju ierīču un sakaru iekārtu ražotājām firmām ASV. Šī firma plaši pazīstama ar mikroprocesoru saimi *Motorola 68000*, ko izmanto *Macintosh* saimes personālajos datoros. Firma *Motorola*, kooperējoties ar firmām *IBM* un *Apple*, izstrādājusi arhitektūru *Power PC* un ražo mikroprocesorus *Power PC 601, 603, 604, 620* u.c. Tos tagad plaši lieto gan personālajos datoros *Power Macintosh*, gan firmas *IBM* datoros. Firma *Motorola* ražo arī datorus *Multi Personal Computer 200*, *Power Slack Risc PC* u.c.

1046 **mouse**
**pele**
**мышь**

Kursora pozicionēšanas ierīce, ar kuru tiek vadīta displeja ekrāna kursora kustība. Peles pārvietošana rada atbilstošu kursora pārvietošanos displeja ekrānā. Pele parasti apgādāta ar divām vai trim vadības pogām, kuru funkcijas atkarīgas no datorā izpildāmās programmas. Informācija no peles datoram tiek nodota pa savienojošo kabeli vai ar infrasarkanajiem stariem. Sk. arī *bus mouse*, *click*, *cordless mouse*, *double click*, *joystick*, *mechanical mouse, optical mouse, optomechanical mouse*..

1047 **mouse button**
**peles poga**
**кнопка мыши**

Konstruktīvs peles elements, kuru nospiežot un atlaižot lietotājs veic kāda objekta izvēli vai iniciē kādas darbības izpildi datorā. Sk. arī *click*, *double-click*.

1048 **mouse pad**
**peles paliktnis**
**подкладка мыши**

Speciāls paliktnis, ko novieto uz galda, lai pa to pārbīdītu peli, nodrošinot tās vieglāku pārvietošanu un precīzāku peles kustības pārnešanu kursora kustībā displeja ekrānā. Sk. arī *mechanical mouse, mouse, optical mouse, optomechanical mouse*.

1049 **mouse pointer**
**peles rādītājs**
**указатель мыши**

Displeja ekrāna elements, kura novietojums mainās, lietotājam pārvietojot peli. Peles rādītājam var būt vairākas formas, kas atkarīgas no tā, kādā režīmā tiek izmantots videoadapteris datorā. Tekstu apstrādes režīmā peles rādītājam ir vertikāla taisnstūra forma un to parasti sauc par bloka kursoru. Grafiku apstrādes režīmā peles rādītājam var būt dažāda forma. Visbiežāk izmantotais tā attēlošanas veids ir pa kreisi ieslīpināta augšupvērsta bultiņa. Sk. arī *pointer*.

1050 **MS-DOS**
**operētājsistēma *MS-DOS*, *MS-DOS***
**операционная система *MS-DOS***

Firmas *Microsoft* izveidota personālo datoru operētājsistēma. Šo operētājsistēmu vienlaicīgi var izmantot tikai viens lietotājs viena uzdevuma risināšanai. Lai organizētu diskus un datnes, *MS-DOS* izmanto kokveida direktoriju struktūru, kur datnes tiek uzglabātas direktorijos un apakšdirektorijos.

1051 **MS-DOS Shell**
**operētājsistēmas *MS-DOS* čaula, *MS-DOS* čaula**
**оболочка операционной системы *MS-DOS***

Uzlabota izvēļņvadāma lietotāju saskarne, kas, sākot ar versiju 5.0, ir operētājsistēmas *MS-DOS* sastāvdaļa. *MS-DOS* čaula nodrošina izvēļņvadāmu pieeju vairumam *MS-DOS* komandu, tādējādi paplašinot šīs operētājsistēmas funkcionalitāti un izmantošanas iespējas.

1052 **MSDOS.SYS**
**datne *MSDOS.SYS***
**файл *MSDOS.SYS***

Viena no divām apslēptajām sistēmas datnēm, kas instalēta *MS-DOS* starta diskā un kurā ir programmatūra operētājsistēmas kodola izveidei. Firmas *IBM* izmantotajās operētājsistēmas *MS-DOS* versijās šo datni sauc par *IBM DOS.COM*.

1053 **MultiColor Graphics Array (MCGA)**
**videostandarts *MCGA*; videoadapteris *MCGA*, adapteris *MCGA***
**адаптер многоцветной графики**

1. Videostandarts, kas izstrādāts *IBM PS/2* modeļiem 25 un 30. Teksta režīmā šī standarta izšķirtspēja 4 krāsām ir 320×400, 2 krāsām — 640×200. Grafikas režīmā 2 krāsām tā augstākā izšķirtspēja ir 640×480, bet 256 krāsām — 320×200.

2. Videoadapteris, kas atbilst videostandartam *MCGA* un, izmantojot piemērotu monitoru, nodrošina šī standarta prasību izpildi, kā arī videostandarta *CGA* emulāciju.

1054 **multifrequency monitor**
**multifrekvenču monitors**
**многочастотный монитор**

Monitors, kas piemērots darbam ar visiem kāda noteikta frekvenču diapazona signāliem. Vairākfrekvenču monitori rada iespēju izmantot dažādus videostandartus.

1055 **multimedia**
**multivide**
**мультимедиа**

Informācijas attēlošana un apstrāde ar vairāk nekā vienas vides palīdzību. Tajā var būt ietverta tekstuālā, grafiskā, audio- un videoinformācija, kā arī animācija. Multivide ir hipervides apakškopa. Hipervide apvieno multivides elementus ar hipertekstu.

1056 **multimedia developer's kit**
**multivides izstrādātāju aprīkojums**
**инструментарий разработчиков мультимедиа**

Firmas *Microsoft* izstrādātā programmatūra, kas ar īpašas programmu bibliotēkas starpniecību ļauj lietotājam izmantot tekstuālo, grafisko un videoinformāciju, veidojot dažādas multivides lietojumprogrammas.

1057 **multimedia extensions**
**multivides paplašinājumi**
**расширения мультимедиа**

Operētājsistēmas papildinājumi, kas nodrošina grafikas un skaņas sinhronizēšanu. Šie paplašinājumi jeb t.s. aizķeres ļauj multivides programmatūras izstrādātājiem izmantot audio- un videoinformāciju bez darbietilpīgu specializētu programmu sastādīšanas.

1058 **multimedia PC**
**multivides personālais dators**
**персональный компьютер мультимедиа**

Personālais dators, kas piemērots multivides lietojumiem un kas parasti ir aprīkots ar skaņu karti, lasāmatmiņas kompaktdiska diskdzini un augstas izšķirtspējas krāsu monitoru.

1059 **multiplexed bus**
**multipleksētā kopne**
**мультиплексная шина**

Kopne, kuras vadu skaits ir mazāks par pārraidāmo adrešu vai datu bitu skaitu. Izmantojot šādas kopnes, informācija tiek pārraidīta laikdales multipleksēšanas režīmā.

1060 **multiplexer**
**multipleksors**
**мультиплексор**

Ierīce, kas ļauj vairākām sakaru līnijām izmantot vienu datu pārraides kanālu.

1061 **multiplexing**
**multipleksēšana**
**мультиплексирование**

Datu pārraides tehnoloģija, kas ļauj diviem vai vairākiem datu avotiem izmantot kopīgu pārraides vidi tā, lai katra datu avota rīcībā tiktu nodots atsevišķs kanāls.

1062 **multiprocessing**
**vairākuzdevumu apstrāde**
**мультиобработка**

1. Vienlaicīga divu vai vairāku programmu vai instrukciju izpilde datorā vai datoru tīklā.
2. Darbu sadale starp vairākiem savstarpēji saistītiem procesoriem.

1063 **Multipurpose Internet Mail Extensions (MIME)**
**tīkla *Internet* pasta vairākmērķu paplašinājumi, paplašinājumi *MIME*, *MIME***
**многоцелевые расширения электронной почты в *Internet***

Paplašinājumu kopa, kas ļauj tīkla *Internet* elektroniskā pasta lietotājiem izmantot savos ziņojumos ne tikai tekstu kodā *ASCII*, bet arī tādus elementus kā grafiku, audio- un videodatnes.

1064 **multitasking**
**vairākuzdevumu režīms**
**многозадачность**

Vienlaicīga divu vai vairāku uzdevumu izpilde vienā datorā.

1065 **multithreading**
**vairākpavedienošana**
**многонитевость**

Laiksakritīga vairāku uzdevumu apstrāde, ko veic viena programma. Tā kā vairākpavedienošanas režīma uzdevumus izpilda paralēli, tad nav nepieciešams sagaidīt viena uzdevuma izpildes pabeigšanu, lai uzsāktu nākošā uzdevuma apstrādi. Sk. arī *thread*.

1066 **multiuser system**
**vairāklietotāju sistēma**
**многопользовательская система**

Datoru sistēma, ko var izmantot vairāki lietotāji, lai vienlaicīgi piekļūtu programmām un datiem. Vairāklietotāju sistēmā katrs lietotājs ir apgādāts ar savu terminālu. Ja sistēmai ir tikai viens centrālais procesors, tad izmanto laikdales režīmu. Par vairāklietotāju sistēmas izveidošanas bāzi var būt personālie

datori, kuros izmantoti mikroprocesori *Intel 80486* un to turpmākās versijas.

1067 **Musical Instrument Digital Interface (MIDI)**
**mūzikas instrumentu ciparsaskarne, saskarne *MIDI, MIDI***
**цифровой интерфейс электромузыкальных устройств**

Standartprotokols, kas nosaka muzikālās informācijas apmaiņu starp mūzikas instrumentiem un datoru. Dators ar šādu saskarni atšķirībā no parastā magnetofona saglabā mūzikas ierakstu nevis analogā, bet gan diskrētā formā, fiksējot taustiņu nospiedienus un vadības kodus.

## N

1068 **name**
**vārds**
**имя**

Rakstzīmju virkne, kas identificē kādu programmas komponentu vai datu struktūru, piem., programmu, datu kopu, procedūru u. tml.

1069 **name server**
**vārdu serveris**
**блок преобразования имен**

Dators, kas ar tīklu *Internet* savienotā lokālajā tīklā nodrošina alfabētisko domēnu vārdu pārveidošanu skaitliskajās starptīklu protokola adresēs (*IP* adresēs). Lai nodibinātu savienojumu ar tīkla *Internet* pakalpojumu sniedzēju, nepieciešams zināt vārdu servera *IP* adresi.

1070 **navigation**
**navigācija**
**навигация**

Hipertekstu pārlūkošanas process, kura gaitā lietotājs, izmantojot hipersaites, izskata savstarpēji saistītus dokumentus, kas var būt izvietoti dažādos datoros. Sk. arī *browser, Netscape Navigator.*

1071 **NCP**

Sk. *NetWare Core Protocol.*

1072 **nested structure**
**ligzdstruktūra**
**вложенная структура**

Sk. *nesting.*

1073 **nesting**
**ligzdošana**
**вложение**

Kādas struktūras iekļaušana citā struktūrā (piem., tabulas iekļaušana datu bāzē; datu struktūras, vadības struktūras vai palīgprogrammas iekļaušana programmā), tādējādi veidojot ligzdtabulas (tabulu tabulā), ligzdprocedūras (procedūras, kas deklarētas procedūrās) vai ligzdierakstus (ierakstus, kas satur lauku, kas pats ir ieraksts).

1074 **NetBIOS**

Sk. *Network Basic Input/Output System.*

1075 **netiquette**
**tīkla etiķete**
**сетевой этикет**

Vispārpieņemtas normas, kas nosaka lietotāja uzvedību, izmantojot tīkla *Internet* pakalpojumus.

1076 **Netscape Navigator**
**pārlūkprogramma**
***Nestcape Navigator*,**
***Netscape Navigator***
**программа ускоренного просмотра *Netscape Navigator***

Globālā tīmekļa *WWW* pārlūkprogramma, ko izstrādājusi firma *Netscape Communications* un ko bez maksas var izmantot ar protokola *FTP* starpniecību. Šī pārlūkprogramma nodrošina ērti izmantojamu saskarni *WWW* lietotājiem un ir ātrdarbīgāka nekā pārlūkprogramma *Mosaic*. Dažkārt šo pārlūkprogrammu sauc vienkārši par *Netscape*.

1077 **NetWare**
**operētājsistēmas**
***NetWare*, *NetWare***
**операционные системы *NetWare***

Firmas *Novell* izstrādātā lokālo tīklu operētājsistēmu saime. Plaši pazīstami tādi šīs saimes produkti, kā, piem., operētājsistēma *Personal NetWare*, kas paredzēta vienādranga tīkliem ar nelielu lietotāju skaitu un operētājsistēmas *NetWare 3.x* un *4.x* — tīkliem ar klient-servera arhitektūru. Sk. arī *Internet Packet Exchange, NetWare Core Protocol, Raster Image Processor, Service Advertising Protocol, Sequenced Packet Exchange.*

1078 **NetWare Core Protocol (NCP)**
**operētājsistēmas *NetWare* kodolprotokols, protokols *NCP***
**протокол ядра *NetWare***

Firmas Novell operētājsistēmas *NetWare* protokols, ko izmanto serveris, lai nodrošinātu *NetWare* klientus ar tīkla pakalpojumiem. Protokola *NCP* rutīnas ļauj izmantot direktorijus un datnes, atvērt semaforus, nodibināt un pārtraukt savienojumus.

1079 **network**
**tīkls**
**сеть**

Datoru un ar tiem saistīto perifērijas ierīču grupa, kas savstarpēji savienotas ar sakaru kanāliem un kas nodrošina datņu un citu resursu kopīgas izmantošanas iespējas vairākiem lietotājiem. Tīklu sarežģītības pakāpe un sniegto pakalpojumu apjoma robežas var būt visai atšķirīgas. Izšķir vienādranga tīklus, kas apvieno nelielu kādas organizācijas lietotāju skaitu, lokālos tīklus, kas, izmantojot kabeļus vai telefona līnijas, savieno lielāku lietotāju skaitu un teritoriālos tīklus, kas apvieno dažādu tīklu lietotājus visai plašā ģeogrāfiskā apgabalā.

1080 **network adapter**
**tīkla adapteris**
**сетевой адаптер**

Sk. *network interface card.*

1081 **Network Basic Input/ Output System (NetBIOS)**
**tīkla ievadizvades pamatsistēma, sistēma *NetBIOS*, *NetBIOS***
**система *NetBIOS***

Lietojumprogrammu un operētājsistēmu (t.sk. *MS-DOS*, *OS/2* un dažu *UNIX* versiju) sadarbības procedūru kopums, kas lokālajam tīklam pievienotajiem personālajiem datoriem nodrošina iespēju izmantot tīkla resursus. Ar īpašas instrukciju kopas starpniecību sistēma

*NetBIOS* dod iespēju lietojumprogrammām pieprasīt zemāko slāņu pakalpojumus, kas nepieciešami, lai organizētu seansus starp tīkla stacijām.

1082 **Network File System (NFS)**
**tīkla datņu sistēma, sistēma *NFS***
**система *NFS***

Firmas *Sun Microsystems* izstrādāta dalīta datņu koplietošanas sistēma, kas ļauj izmantot attāla datora datnes un perifērijas iekārtas tā, it kā tās atrastos lietotāja datorā. Sistēma *NFS* nav atkarīga no datora procesora tipa un operētājsistēmas, kā arī izmantojamā tīkla arhitektūras un protokoliem. Šī sistēma iekļauta daudzu ražotāju piegādātajā programmatūrā un tiek plaši izmantota tīklā *Internet*.

1083 **Network Interface Card (NIC)**
**tīkla saskarnes karte, karte *NIC***
**сетевая интерфейсная плата**

Personālā datora izvērses karte, kas kopā ar tīkla operētājsistēmu vada informācijas plūsmu datoru tīklā. Tīklam darbojoties, šī karte ir tieši saistīta ar datu pārraides vidi (vīto pāri, koaksiālo vai optisko kabeli), kas, savukārt, saista savā starpā visas tīkla saskarnes kartes. Dažkārt šo karti sauc arī par tīkla adapteri.

1084 **newline character**
**jaunās rindiņas rakstzīme**
**знак новой строки**

Vadības rakstzīme, kas norāda, ka kursoram ekrānā vai printera drukājošajam mehānismam jāpārvietojas uz nākošās rindiņas sākumu. Jaunās rindiņas rakstzīme funkcionāli ir ekvivalenta rakstatgriezes un rindpadeves rakstzīmju apvienojumam.

1085 **newsgroup**
**intereškopa**
**группа новостей**

Ziņojumdēļa sistēmas lietotāju grupa, kuru apvieno kopīgas intereses, piem., sports, mūzika, vēsture vai politika.

1086 **newsreader**
**jaunumlasītājs**
**программа чтения новостей**

Lietojumprogrammatūra intereškopām nosūtītās informācijas lasīšanai.

1087 **NFS**

Sk. *Network File System*.

1088 **NIC**

Sk. *Network Interface Card*.

1089 **node**
**mezgls**
**узел**

Jebkura datoru tīklam pievienota ierīce, kas var veikt informācijas apmaiņu ar citām datoru tīklam pievienotajām ierīcēm.

1090 **non-volatile memory**
**energoneatkarīga atmiņa**
**энергонезависимая память**

Atmiņa, kas izveidota tā, lai saglabātu ierakstīto informāciju arī tad, kad barošanas spriegums tiek atslēgts. Energoneatkarīga, piem., ir lasāmatmiņa. Sk. arī *volatile memory*.

1091 **nondestructive read**
**nedestruktīvā lasīšana**
**считывание без разрушения**

Datu nolasīšanas operācija, kuru izpildot nolasītie dati tiek saglabāti datora atmiņā. Nedestruktīvo nolasīšanu nodrošina īpaša atmiņas izveides tehnoloģija vai nolasīšanas operācijai sekojoša datu atsvaidzināšana.

1092 **noninterlacing**
**rindsecīgā izvērse**
**построчная развертка**

Monitoros izmantojama attēla izspīdināšanas tehnoloģija, kas ar elektronu staru, vienā caurgājienā monitora ekrānā zīmē pēc kārtas visas līnijas. Rindsecīgo izvērsi parasti pielieto datorgrafikā izmantojamajos monitoros, jo tā dod blīvu attēlu un novērš ekrāna mirgoņu. Sk. arī *interlacing*.

1093 **Norton Desktop for Windows**
**lietojumprogramma *Norton Desktop for Windows*, *Norton Desktop for Windows***
**прикладная программа *Norton Desktop for Windows***

Lietojumprogramma, kas paredzēta izmantošanai operētājsistēmas *Microsoft Windows* vidē un kas apvieno lietojumprogrammu *Program Manager* un *File Manager* funkcijas. Tā nodrošina dažādus pakalpojumus (t.sk. datņu atkopšanu un dublēšanu), kā arī līdzekļus ar disku diagnosticēšanu saistītu problēmu risināšanai.

1094 **notebook computer**
**piezīmjdators**
**блокнотный компьютер**

Portatīvs personālais dators ar autonomu barošanas avotu, nelielu tastatūru un displeju. Pēc saviem izmēriem piezīmjdators ir mazāks par klēpjdatoru un lielāks par kabatdatoru. Datu ievadīšanai piezīmjdatorā tastatūras vietā dažkārt izmanto gaismas zīmuli.

1095 **Novell NetWare**

Sk. *NetWare*.

1096 **Novell, Inc.**
**firma *Novell***
**фирма *Novell***

Viena no ASV vadošajām firmām datoru tīklu programmatūras izstrādāšanas jomā. Firma *Novell* 1986. g. izstrādāja programmatūru protokola *TCP/IP* uzturēšanai personālajos datoros. Šī firma izstrādājusi personālo datoru lokālo tīklu operētājsistēmas *NetWare* un *Personal Netware*.

1097 **Nu Bus**
**kopne *Nu Bus***
**шина *Nu Bus***

70. gadu beigās Masačusetas Tehnoloģiskajā institūtā izstrādāta ātrdarbīga 32 bitu izvēršanas kopne ar centralizētu arbitrāžu. Kopne *Nu Bus* vēlāk tika izmantota firmas *Apple Computer Inc. Macintosh* saimes personālajos datoros.

1098 **Num Lock key (Numeric Lock key)**
**ciparslēga taustiņš**
**клавиша переключения и фиксации регистра вспомогательной клавиатуры**

Tastatūras taustiņš, kas darbojas kā pārslēgs. Ar ieslēgtu ciparslēga taustiņu cipartatstatūras taustiņus var izmantot skaitlisku datu ievadei, tāpat kā strādājot ar kalkulatoru. Ar atslēgtu ciparslēga taustiņu lielākā daļa cipartastatūras taus-

tiņu tiek izmantoti kursora vadībai un ekrāna ritināšanai.

1099 **numeric coprocessor**
**skaitliskais līdzprocesors**
**цифровой сопроцессор**
Sk. *math coprocessor*.

1100 **numeric keypad**
**cipartastatūra**
**цифровая клавиатура**
Tastatūras taustiņu grupa, kas paredzēta ātrai skaitlisko datu ievadei datorā. Cipartastatūra parasti izvietota firmas *IBM* un ar to saderīgo tastatūru labajā pusē.

1101 **Object Linking and Embedding (OLE)**
**objektu sasaiste un iegulte, standarts *OLE*, *OLE***
**компоновка и внедрение объектов**
Standartu kopa, ko izstrādājusi firma *Microsoft* un kas izmantota operētājsistēmās *Microsoft Windows* un firmas *Apple Macintosh* saimes datoros. Šie standarti tiek lietoti, lai izveidotu dinamiskas, automātiski atjaunināmas saites starp dokumentiem, kā arī lai dokumentu, kas radīts vienam lietojumam, ievietotu dokumentā, kas izveidots citam lietojumam.

1102 **object-oriented graphics**
**objektorientēta grafika**
**объектно-ориентированная графика**
Sk. *vector graphics*.

1103 **OCR**
Sk. *optical character recognition*.

1104 **off-line mode**
**nesaistes režīms**
**автономный режим**
Datu apstrādes sistēmas funkcionālā bloka darbības režīms, kurā aplūkojamais funkcionālais bloks strādā neatkarīgi no tā, vai tas ir pieslēgts sistēmai vai arī atslēgts no tās. Sistēma tā darbību nevada.

1105 **office automation**
**biroja automatizācija**
**автоматизация учрежденческой деятельности**
Datoru un datoru tīklu izmantošana birojā, lai uzlabotu tā darba produktivitāti un vadības efektivitāti, izmantojot, piem., elektroniskās kartotēkas, tekstu apstrādes sistēmas, datorgrafiku, elektronisko pastu un telekonferences. Sk. arī *Microsoft Office*.

1106 **offset**
**nobīde**
**смещение; отступ**
1. Skaitlis, kas norāda attālumu no datnes vai atmiņas adrešu sākumpunkta. Pieskaitot šo skaitli bāzes adresei, iegūst absolūto adresi. Datnēs šis skaitlis norāda rakstzīmes (baita) novietojumu, skaitot no datnes sākuma.
2. Teksta apstrādē nobīde rāda dokumentā nodrukātā teksta attālumu no tā kreisās malas.

1107 **OK button**
**apstiprinājumpoga, poga "Labi"**
**кнопка подтверждения**
Spiežampoga grafiskās lietotāja saskar-

nes dialoglodziņā, ar kuras starpniecību lietotājs apstiprina savu izvēli un inicie komandas izpildi.

1108 **OLE**
Sk. *Object Linking and Embedding.*

1109 **on-line help**
**tiešsaistes palīdzība**
**оперативная помощь**
Displeja ekrānā pieejama palīdzība, ko lietotājs var izmantot, strādājot ar datoru tīklu vai lietojumprogrammu.

1110 **on-line mode**
**tiešsaistes režīms**
**оперативный режим**
Datu apstrādes sistēmas funkcionālā bloka darbības režīms, kura būtiskā pazīme ir tā, ka šis bloks sadarbībā ar citām sistēmas daļām strādā tiešā sistēmas vadībā.

1111 **on-screen formating**
**ekrānformatēšana**
**непосредственное форматирование**
Tekstapstrādes programmās — formatēšanas tehnika, kas nodrošina formatēšanas komandu tiešu izmantošanu displeja ekrānā redzamā teksta rediģēšanai. Sk. arī *WYSIWYG.*

1112 **open architecture**
**atvērtā arhitektūra**
**открытая архитектура**
Arhitektūra, kuras specifikācija ir publiski pieejama un kuru brīvi var izmantot citi izgatavotāji savas produkcijas ražošanai. Tā, piem., *IBM PC* arhitektūra ir atvērta un to izmanto citi ražotāji. Sk. *closed architecture.*

1113 **operating system**
**operētājsistēma**
**операционная система**
Programmu komplekss, kas vada datu organizēšanu un programmu izpildi datorā, nodrošina aparatūras un programmatūras kopdarbību, resursu racionālu izmantošanu, kā arī sadarbību ar lietotāju. Pazīstamākās personālo datoru operētājsistēmas ir *MS-DOS*, *OS/2*, *Microsoft Windows 95* un *UNIX*. Sk. arī *disk operating system.*

1114 **optical character recognition**
**rakstzīmju optiskā pazīšana**
**оптическое распознавание символов**
Drukātu rakstzīmju atpazīšana datorā. Optiskā rakstzīmju pazīšanas sistēma var atpazīt daudz dažādu fontu, tai skaitā mašīnraksta un datora izdrukas fontus. Attīstītas optiskās rakstzīmju pazīšanas sistēmas var atpazīt pat ar roku rakstītu tekstu.

1115 **optical disc**
**optiskais disks**
**оптический диск**
Lielas ietilpības datoru atmiņas ierīce, kurā informāciju ieraksta un nolasa ar lāzera stara palīdzību. Ierakstītās informācijas blīvums optiskajos diskos ir ievērojami augstāks nekā magnētiskajos diskos, bet tās nolasīšanas ātrums — mazāks. Sk. arī *CD-ROM, erasable optical disc, magneto-optical disc, WORM.*

1116 **optical mouse**
**optiskā pele**
**оптическая мышь**
Pele, kas par savu atrašanās vietu saņem

informāciju, uztverot pašas izstaroto gaismu, kas atstarojas no speciāla paliktņa ar precīzu koordinātu režģi. Optiskajā pelē ir mazāk kustīgu detaļu nekā mehāniskajā vai optomehāniskajā pelē, tāpēc tā ir drošāka, bet, tā kā šī tipa pelei ir nepieciešams speciāls paliknis, tā ir dārgāka.

1117 **optical reader**
**optiskais lasītājs**
**оптическое читающее устройство**

Ierīce drukāta teksta un svītrkodu lasīšanai, kodēšanai un ievadīšanai datorā. Optiskais lasītājs sastāv no skenera teksta lasīšanai un programmatūras nolasīto rakstzīmju analīzei.

1118 **option key**
**opcijas taustiņš**
**клавиша выбора**

Firmas *Apple* tastatūras taustiņš, kurš apvienojumā ar kādu rakstzīmes taustiņu veido speciālus simbolus, piem., lodziņus, dažādu valūtu simbolus, slīpsvītras u.c. Opcijas taustiņa funkcijas ir līdzīgas vadīšanas vai alternēšanas taustiņa funkcijām firmas *IBM* un ar to saderīgajās tastatūrās.

1119 **optomechanical mouse**
**optomehāniskā pele**
**оптомеханическая мышь**

Pele, kurā atšķirībā no mehāniskās peles lodītes kustība tiek pārveidota elektriskajos signālos, izmantojot kā mehāniskus konstruktīvus elementus, tā arī gaismu emitējošas diodes un fototranzistorus. Atšķirībā no optiskās peles optomehāniskās peles izmantošanai nav nepieciešams speciāls paliktnis.

1120 **Oracle**
**datu bāzu pārvaldības sistēma *Oracle*, sistēma *Oracle*, *Oracle***
**система управления базой данных *Oracle***

Firmas *Oracle Corp.* relāciju datu bāzu pārvaldības sistēma, ko izmanto plašam datoru spektram — no mikrodatoriem līdz pat lieldatoriem. Tā bija viena no pirmajām datu bāzu pārvaldības sistēmām, kas izmantoja valodu *SQL*.

1121 **orphan [line]**
**bāreņrindiņa**
**висячая строка**

Teksta apstrādē pirmo paragrāfa rindiņu sauc par bāreņrindiņu, ja tā parādās viena pati lappuses apakšā. Vairums teksta apstrādes un izklājlapu programmu nepieļauj šādu atsevišķu rindiņu parādīšanos.

1122 **OS/2**
**operētājsistēma *OS/2*, *OS/2***
**операционная система *OS/2***

Vairākuzdevumu operētājsistēma, ko izmanto firmas *IBM* un ar tiem saderīgajos personālajos datoros un kas vienlaicīgi nodrošina izpildāmo programmu aizsardzību, kā arī ļauj veikt dinamisku datu apmaiņu starp lietojumprogrammām. Operētājsistēma *OS/2* nodrošina praktiski visu operētājsistēmai *MS-DOS* rakstīto programmu izpildi un ļauj lasīt diskatmiņā ierakstīto informāciju, kas nepieciešama šo programmu izpildei.

1123 **outline font**
**kontūrfonts**
**контурный шрифт**

Fonts, kuru ar matemātisku formulu

palīdzību veido kā katras rakstzīmes pamatkontūru. Pirms drukāšanas šos kontūrus var mērogot un pārveidot par bitkartētām rakstzīmēm. Vairākiem lāzerprinteru tipiem ir iebūvēti kontūrfonti. Sk. arī *scalable font, screen font.*

1124 **output**
**izvade**
**вывод**

Datorā izveidotās informācijas izspīdināšana displeja ekrānā, izdrukāšana, ierakstīšana diskā (lentē) vai nosūtīšana citam datoram ar sakaru līdzekļu starpniecību. Sk. arī *input, input/output.*

1125 **output device**
**izvadierīce**
**устройство вывода**

Ierīce datu pārveidošanai no formas, kādā tie glabājas datora atmiņā, formā, kādu var lietot ārpus datu apstrādes sistēmas. Sk. arī *display, plotter, printer.*

1126 **overflow**
**pārpilde**
**переполнение**

Noteiktu datu vai programmu uzglabāšanai datorā paredzētā atmiņas apjoma pārsniegšana.

1127 **overlaid windows**
**pārklājlogi**
**перекрывающиеся окна**

Sk. *cascading windows.*

1128 **overwrite mode**
**pārraksta režīms**
**режим замены**

Programmas darbības režīms, kurā teksts, ko ieraksta dokumentā vai komandu rindā, tiek rakstīts virsū pa labi no kursora esošajam tekstam. Pārraksta režīmam pretējs ir iespraušanas režīms.


1129 **PA**

Sk. *Precision Architecture.*

1130 **packet**
**pakete**
**пакет**

Nedalāms datu bloks, ko pārsūta pa datu pārraides tīklu. Paketē parasti ir ne tikai lietotāja dati, bet arī informācija par datu nosūtītāju un datu saņēmēju, kā arī informācija kļūdu atklāšanai. Paketes garums var būt fiksēts vai mainīgs. Pakešu komutācijas tīklos parasti tiek noteikts maksimālais pieļaujamais paketes garums.

1131 **pad character**
**pildījuma rakstzīme**
**символ-заполнитель**

Rakstzīme, kuru izmanto tukšo vietu aizpildīšanai fiksēta garuma datu blokos.

1132 **padding**
**papildinājums**
**наполнение**

Datu bloku papildināšana ar fiktīvām rakstzīmēm, vārdiem vai ierakstiem, līdz tie sasniedz iepriekš noteikto datu bloka garumu.

1133 **page**
**lappuse**
**страница**

1. Fiksēta garuma brīvpieejas atmiņas bloks.
2. Tekstapstrādē vai datorizdevnie-

cībā — drukājamās lappuses vai grafikas attēls displeja ekrānā.

**1134 page break**
**lappuses pārtraukums**
**разбиение страницы**

Tekstapstrādē — kods, kas norāda, ka jābeidz teksta drukāšana kārtējā lappusē un jāpāriet uz jaunu lappusi. Vairums tekstapstrādes programmu automātiski pārrauj lappusi, kad tā ir pilna. Šādu pārtraukumu sauc par nestingro lappuses pārtraukumu. Lietotājs var ievadīt arī stingro lappuses pārtraukumu, kurš piespiež programmu sākt drukāt jaunā lappusē no tās vietas, kur ir fiksēts stingrais lappuses pārtraukums.

**1135 page description language**
**lappuses aprakstvaloda**
**язык описания страницы**

Augsta līmeņa valoda printera izdrukas aprakstīšanai. Ar šīs valodas palīdzību ievērojams printera izvada veidošanas darba apjoms no datora tiek pārnests uz printeri. Piem., tā vietā, lai pārlādētu visu fontu no datora uz printeri, printerim tiek nosūtīta komanda veidot konkrētu fontu ar noteiktu punktu izmēru, un printeris no fonta pamatelementiem (kontūriem) izveido fonta rakstzīmes.

**1136 Page Down key (Pg Dn)**
**lejupšķiršanas taustiņš**
**клавиша листания вниз**

Taustiņš, kas apvienots ar taustiņu 3 un izvietots cipartastatūras daļā, kā arī paplašinātās tastatūras rediģēšanas taustiņu blokā starp tastatūras pamatdaļu un cipartastatūru. Lapas lejupšķiršanas taustiņam dažādās programmās mēdz būt atšķirīgas funkcijas. Visbiežāk to izmanto tekstapstrādes programmās, lai dokumentu ritinātu par viena ekrāna saturu uz leju.

**1137 page fault interrupt**
**lappuses iztrūkumpārtraukums**
**прерывание по отсутствию страницы**

Pārtraukums, ko izsauc programmatūra, kas mēģina lasīt no virtuālās atmiņas vai ierakstīt tajā, kaut gan attiecīgais virtuālās atmiņas apgabals šajā brīdī neatrodas fiziskajā atmiņā.

**1138 page layout**
**lappuses izkārtojums**
**компоновка страницы**

Teksta vai grafikas izvietojums kāda dokumenta lappusē. Lappuses izkārtojumu parasti veic īpašas programmas, ar kurām tekstu gan izvieto, gan apstrādā. Sk. arī *page layout program*.

**1139 page layout program**
**lappuses**
**izkārtojumprogramma**
**программа компоновки страницы**

Datorizdevniecības lietojumprogramma, kas palīdz apvienot tekstu un grafiku no dažādām datnēm vienotā dokumentā, kā arī veikt ar šī dokumenta tekstu un attēliem virkni operāciju: teksta izkārtošanu vairākās slejās, teksta izvietojumu ap grafikas attēliem, teksta apcirpšanu, attēlu mērogošanu u.c. Sk. arī *Page Maker, Ventura Publisher*.

**1140 page makeup**
**lappuses izveide**
**верстка страницы**

Drukājamās lappuses formēšana, kas ietver galveņu, kājeņu, kolonnu, lappuses numura, robežsvītru, grafikas un teksta izvietošanu.

1141 **page preview**
**lappuses priekšskatījums**
**предварительный просмотр страницы**

Ar tekstapstrādes un datorizdevniecības programmatūru veidots precīzs grafisks lappuses attēlojums, izmantojot paredzēto burtstilu, malu un grafikas izvietojumu un tādējādi ļaujot displeja ekrānā pilnībā novērtēt, kā izdrukā izskatīsies veidojamā lappuse.

1142 **page setup**
**lappuses uzstādīšana**
**установка страницы**

Programmatūras papildiespējas, kas ļauj izveidot lappusi tādu, kāda tā izskatīsies izdrukā, piem., noteikt tās lielumu, malu platumu, papīra izmēru. Sk. arī *page preview*.

1143 **Page Up key (Pg Up)**
**augšupšķiršanas taustiņš**
**клавиша листания вверх**

Taustiņš, kas apvienots ar taustiņu 9 un izvietots firmas *IBM* un ar to saderīgas tastatūras cipartastatūras daļā, kā arī paplašinātās tastatūras rediģēšanas taustiņu blokā starp tastatūras pamatdaļu un cipartastatūru. Lapas augšupšķiršanas taustiņam dažādās programmās mēdz būt atšķirīgas funkcijas. Visbiežāk to izmanto tekstapstrādes programmās, lai dokumentu ritinātu par viena ekrāna saturu uz augšu.

1144 **PageMaker**
**datorizdevniecības programma *PageMaker*,**
***PageMaker***
**издательская программа *PageMaker***

Izplatīta datorizdevniecības programma, kas ļauj integrēt tekstu un grafiku un ko izmanto rakstu, pārskatu, reklāmu, apkārtrakstu un citu dokumentu sagatavošanai. Šī programma paredzēta *Macintosh* saimes un firmas *IBM* vai ar tiem saderīgiem datoriem.

1145 **pagination**
**lapdale**
**разбиение текста на страницы**

Dokumenta sadalīšana lappusēs un to attēlošana displeja ekrānā pirms izdrukāšanas. Lietotājs var novērtēt dokumenta noformējumu pa sadaļām un veikt vēl citus nelielus labojumus.

1146 **paging**
**lapošana**
**листание**

1. Grafikas vai teksta lappuses attēlošana displeja ekrānā.
2. Virtuālās atmiņas izveidošanas metode, pārsūtot virtuālās programmu lappuses fiziskajā atmiņā un no fiziskās atmiņas virtuālajā atmiņā. Speciāla atmiņas pārvaldības aparatūra translē virtuālās adreses fiziskajās adresēs.

1147 **paint program**
**krāsošanas programma**
**программа рисования**

Datorgrafikas programma attēlu zīmēšanai displeja ekrānā, veidojot zīmējumu no atsevišķiem punktiem vai pikseļiem ar grafiskās planšetes vai peles palī-

dzību. Atšķirībā no zīmēšanas programmas krāsošanas programma izmanto rastrgrafiku, nevis vektorgrafiku. Sk. arī *bit mapped graphics*.

1148 **palette**
**palete**
**палитра**
Krāsu kopa, kas pieejama zīmēšanas vai krāsošanas programmām. Paletē var ietilpt ne tikai krāsas, bet arī šabloni, otu veidi un dažāda biezuma līnijas.

1149 **palmtop computer**
**plaukstdators**
**ладонный компьютер**
Mazs dators, kas novietojams plaukstā. Salīdzinot ar normāla lieluma datoriem, tā iespējas ir ierobežotas, bet šie datori ir ērti izmantojami, piem., telefonu sarakstu, kalendāru un citu līdzīga rakstura datu uzglabāšanai un apstrādei. Kā barošanas avotu plaukstdatoros parasti izmanto plaši pieejamas baterijas.

1150 **panning**
**panoramēšana**
**панорамирование**
Metode, ko izmanto datorgrafikā, lai aplūkotu displeja ekrānā kādu liela atmiņā uzglabājama attēla daļu, ko panāk, pārvietojot horizontāli vai vertikāli panorāmas logu (skatu meklētāju).

1151 **Pantone Matching System (PMS)**
**pantoņu saskaņošanas sistēma, sistēma *PMS*, *PMS***
**система соответствия цветов**
Krāsu pieskaņošanas standartsistēma krāsu attēlu veidošanai displeja ekrānā vai krāsainu ilustrāciju izdrukāšanai ar strūklprinteri, izmantojot katalogu, kurā ir ap 500 krāsu paraugu.

1152 **paper-white display**
**papīrbalts displejs**
**бумажно-белый монитор**
Sk. *paper-white monitor*.

1153 **paper-white monitor**
**papīrbalts monitors**
**бумажно-белый монитор**
Augstas kvalitātes monohroms monitors, kas attēlo melnu tekstu vai grafiku uz balta fona. Šī tipa monitorus parasti izmanto tekstapstrādes vai datorizdevniecības uzdevumu risināšanai, jo to attēls pēc izskata ir ļoti līdzīgs drukātai lappusei.

1154 **paragraph**
**rindkopa**
**абзац**
Dokumenta daļa, kas tekstapstrādes procesā sākas ar vienu speciālu rindkopas zīmi un beidzas ar otru. Parasti tekstapstrādes programmas rindkopu uztver kā informācijas bloku, ko var atlasīt un apstrādāt atsevišķi no apkārtējām rindkopām.

1155 **parallel port**
**paralēlā pieslēgvieta**
**параллельный порт**
Ievadizvades pieslēgvieta, kas nodrošina informācijas ātru sinhronu pārsūtīšanu pa paralēlām līnijām starp datoru un ārējām ierīcēm (piem., paralēlo printeri). Sk. arī *serial port*.

1156 **parallel printer**
**paralēlais printeris**
**параллельный принтер**
Printeris, ko paredzēts pieslēgt datora

paralēlajai pieslēgvietai. Paralēlo printeri var novietot tālāk no datora un parasti to ir vieglāk instalēt un lietot nekā seriālo printeri.

1157 **park**
**galviņievilkšana**
**позиционирование головок**
Cietā diska galviņu fiksēšana pozīcijā, kas izslēdz plates virsmas bojāšanos, diskdzini pārvietojot. Vairumam moderno diskdziņu, izslēdzot barošanas spriegumu, galviņas fiksējas automātiski.

1158 **parser**
**parsētājs**
**синтаксический разделитель**
Programma, kas lielus datu blokus sadala mazākās, vieglāk interpretējamās daļās.

1159 **partition**
**nodalījums**
**раздел**
Cietā diska atmiņas apgabala daļa, kas izveidota tā, ka operētājsistēma to uztver kā atsevišķu diskatmiņu. Sk. arī *logical drive.*

1160 **passive hub**
**pasīvais centrmezgls**
**пассивный концентратор**
Centrmezgls, ko izmanto zvaigžņtīklos, lai pārraidāmos signālus sadalītu starp darbstacijām un citiem centrmezgliem. Pasīvais centrmezgls atšķirībā no aktīvā centrmezgla nepastiprina tīklā pārraidāmos signālus.

1161 **passive matrix display**
**pasīvās matricas displejs**
**дисплей на жидких кристаллах с пассивной матрицей**
Šķidro kristālu displejs, kura izveidošanas tehnoloģija nodrošina attiecīgā pikseļa izgaismošanu, padodot strāvu atbilstošajai rindai un kolonnai. Sk. arī *active matrix display.*

1162 **password**
**parole**
**пароль**
Rakstzīmju (parasti burtu un ciparu) virkne, ko izmanto, lai identificētu datora sistēmas programmas vai tīkla lietotāju, kuram ir tiesības izmantot datora sistēmas programmu vai tīklu. Dažkārt parole precizē izmantošanas tiesības, nosakot, piem., vai lietotājs drīkst tikai lasīt informāciju, vai lasīt un ierakstīt datus vai arī kopēt datnes.

1163 **paste**
**ielīmēt**
**вставить**
Teksta, grafikas, datu bāzes ieraksta vai cita informācijas bloka iespraušana dokumentā, ko veic, šo bloku iepriekš izgriežot no teksta vai nokopējot starpliktuvē un pēc tam ievietojot noteiktā gatavojamā dokumenta vietā. Sk. arī *block move, cut and paste.*

1164 **patch**
**ielāps**
**заплата**
Neliela programmas korekcija, ko parasti veic lietotājs, lai novērstu iepriekš neparedzētus trūkumus programmas funkcionēšanā.

1165 **path**
**ceļš; trakts**
**путь; тракт**
1. Orientēta grafa šķautņu secība, kas

savieno divas grafa virsotnes tā, ka šķautņu virzieni ir savstarpēji saskaņoti.
2. Maršruts starp jebkuriem diviem tīkla mezgliem.
3. Komandas, signāla vai datu pārsūtīšanas maršruta no datu avota līdz datu saņēmējam fiziskā realizācija (līnija, kanāls, posms vai ķēde).
4. Maršruts, kas operētājsistēmai jānoiet, lai atrastu izpildāmo programmu apakšdirektorijā.

1166 **pathname**
**ceļa vārds**
**имя пути**

Operētājsistēmas *MS-DOS* priekšraksts, kas norāda datnes vārdu un tās precīzu atrašanās vietu uz cietā diska. Ja lietojumprogrammās jāatver vai jāsaglabā datne, kas atrodas aktuālajā direktorijā, tad, lai to atrastu vai saglabātu, jāuzrāda visu direktoriju hierarhija.

1167 **Pause key**
**pauzēšanas taustiņš**
**клавиша паузы**

Firmas *IBM* un ar to saderīgas tastatūras taustiņš, kas uz laiku pārtrauc programmas vai komandas izpildi. Tā, piem., pauzēšanas taustiņš tiek izmantots, lai pārtrauktu gara teksta izvadi uz ekrāna un ļautu lietotājam to izlasīt.

1168 **PC (Personal Computer)**
**personālais dators *PC*, *PC***
**персональный компьютер *PC***

Tādu personālo datoru apzīmējums, kas atbilst firmas *IBM* izstrādāto personālo datoru standartam un ir ar tiem saderīgi. Šo apzīmējumu izmanto arī firmas *IBM* personālajiem datoriem *PC/XT* un *PC/AT*, lai tos atšķirtu no otrās paaudzes datoriem *IBM PS/2*.

1169 **PC bus**
**kopne *PC***
**шина *PC***

Firmas *IBM* pirmās paaudzes personālo datoru datu kopne. Šo apzīmējumu parasti attiecina uz 8 bitu kopni un tās 16 bitu paplašinājumu, ko nodrošina tehnoloģija *AT*. Šī tipa kopnes sauc arī par *ISA* kopnēm. Sk. arī *AT bus, Industrial Standard Architecture.*

1170 **PC Card**
Sk. *PCMCIA card.*

1171 **PC Card slot**
Sk. *PCMCIA slot.*

1172 **PC-DOS**
**operētājsistēma *PC-DOS*, *PC-DOS***
**операционная система *PC-DOS***

Operētājsistēma, kuru firma *IBM* izveidojusi saviem datoriem. *PC-DOS* un *MS-DOS*, ir funkcionāli identiskas operētājsistēmas. Šīs sistēmas dažkārt atšķiras tikai ar datņu un utilītu nosaukumiem.

1173 **PC/AT**
**personālais dators *PC/AT*, *PC/AT***
**персональный компьютер *PC/AT***

Firmas *IBM* personālais dators, kurā izmantots 16 bitu *Intel 80286* mikroprocesors un paplašināšanas kopne, kas izstrādāta atbilstoši arhitektūras *ISA* prasībām. Sk. arī *Advanced Technology.*

1174 **PC/XT**
**personālais dators *PC/XT*, *PC/XT***
**персональный компьютер *PC/XT***

Firmas *IBM* personālais dators, kas izstrādāts, izmantojot pirmā firmas *IBM* personālā datora arhitektūru, un papildināts ar arhitektūrai *ISA* atbilstošu paplašināšanas kopni un cieto disku. Sk. arī *eXtended Technology.*

1175 **PCI**
Sk. *Peripheral Component Interconnect.*

1176 **PCI local bus**
**lokālā kopne *PCI***
**локальная шина *PCI***
Firmas *Intel* izstrādāta augstas caurlaidspējas 32 bitu lokālā kopne. Lokālā kopne *PCI*, izmantojot multipleksēšanu, nodrošina 64 bitu datu pārraidi, bet sprādzienrežīmā — neierobežota garuma datu masīvu pārraidi. Lokālās kopnes *PCI* saiti ar arhitektūras *ISA, EISA, MCA* kopnēm, kā arī ar kopni *NuBus* nodrošina kopņu tilti. Sk. arī *bus mastering, extension bus, Peripheral Component Inerconnect.*

1177 **PCI slot**
***PCI* slots**
**слот *PCI***
Adapteru ligzda mātesplatē, kas aprīkota ar specifikācijai *PCI* atbilstošu paplašināšanas kopni. *PCI* sloti, salīdzinot ar slotiem, ko izmanto *VL* kopnēs un arhitektūras *ISA* kopnēs, ir ātrdarbīgāki un tiek izmantoti, piem., mātesplatēs ar *Pentium* mikroprocesoriem. Sk. arī *Industry Standard Architecture, VL-bus.*

1178 **PCMCIA**
Sk. *Personal Computer Memory Card International Association.*

1179 **PCMCIA card**
**karte *PCMCIA***
**карта *PCMCIA***
Atmiņas vai perifērijas ierīces karte, kas atbilst *PCMCIA* standartam.

1180 **PCMCIA slot**
**slots PCMCIA**
**слот PCMCIA**
Portatīvā datora ligzda, kas izveidota *PCMCIA* standartiem atbilstošu karšu ievietošanai. Sk. *PC Card, PCMCIA card.*

1181 **PCX**
**formāts *PCX*, *PCX***
**формат *PCX***
Datnes standartformāts krāsainu grafikas attēlu uzglabāšanai datora atmiņā.

1182 **PDA**
Sk. *Personal Digital Assistant.*

1183 **peer-to-peer architecture**
**vienādranga arhitektūra**
**равноранговая архитектура**
Datoru tīkla uzbūves koncepcija, kas paredz, ka visām stacijām ir vienādas datu pārraides tiesības. Šāda tipa arhitektūra izmantota, piem., atvērto sistēmu sadarbības bāzes etalonmodelī. Sk. arī *client/server architecture, peer-to-peer network.*

1184 **peer-to-peer network**
**vienādranga tīkls**
**равноранговая сеть**
Lokālais tīkls, kurā katra personālā datora diskdziņi, datnes un printeri ir pieejami no jebkura cita šim tīklam pieslēgtā datora. Šādas arhitektūras tīklos īpaša datņu servera izmantošana nav obligāta.

1185 **PEM**
Sk. *Privacy Enhanced Mail.*

1186 **pen computer**
**zīmuļievades dators**
**компьютер с рукописным вводом**

Datori, kuru primārā ievadierīce ir speciāls zīmulis (irbulis), nevis tastatūra. Šīs klases datoriem ir plakans ekrāns un speciāla operētājsistēma, kas nodrošina lietotāja sadarbību ar datoru.

1187 **Pentium**

Sk. *Intel Pentium*.

1188 **Pentium Pro**

Sk. *Intel Pentium Pro*.

1189 **perfective maintenance**
**uzlabojošā uzturēšana**
**усовершенствованное сопровождение**

Programmatūras uzturēšana, kas nodrošina tās veiktspējas vai citu raksturojumu uzlabošanu.

1190 **performance**
**veiktspēja**
**производительность**

Sistēmas vai tās komponentu spēja izpildīt paredzētās funkcijas. Kā veiktspējas kvantitatīvie kritēriji parasti tiek izmantoti atbildes laiks, caurlaidspēja un izmantojamība. Sk. arī *performance testing*.

1191 **performance testing**
**veiktspējas testēšana**
**тестирование производительности**

Testēšana, kas novērtē sistēmas atbilstību specifikācijā noteiktajām veiktspējas prasībām.

1192 **Peripheral Component Interconnect (PCI)**
**specifikācija *PCI***
**спецификация *PCI***

Firmas *Intel* izstrādāta ātrdarbīgas lokālās kopnes specifikācija, kas nodrošina ātru datu pārsūtīšanu starp procesoru un šai specifikācijai atbilstošām izvērses platēm.

1193 **permanent font**
**permanentfonts**
**резидентный шрифт**

Fonts, kas pēc ielādēšanas lāzerprinterī saglabājas tā atmiņā tik ilgi, kamēr printeris pieslēgts barošanas avotam. Sk. arī *downlodable font, temporary font*.

1194 **personal computer**
**personālais dators**
**персональный компьютер**

Vispārīgas lietošanas dators, kas paredzēts individuālai izmantošanai, piem., biroja darbā, tirdzniecībā, sadzīvē u.c. Sk. arī *desktop computer, home computer, laptop computer, microcomputer, notebook computer, palmtop computer, pocket computer, portable computer*.

1195 **Personal Computer Memory Card International Association (PCMCIA)**
**Starptautiskā personālo datoru atmiņas karšu asociācija, asociācija *PCMCIA***
**Международная ассоциация карт памяти персональных компьютеров, ассоциация *PCMCIA***

Starptautiska ražotāju un tirdzniecības

organizāciju asociācija, kas izstrādā atmiņas karšu un attiecīgo slotu standartus portatīviem datoriem. Sk. *PC Card, PC Card slot, PCMCIA card, PCMCIA slot.*

1196 **Personal Digital Assistant (PDA)**
**personālais ciparasistents**
**персональный цифровой помощник**

Plaukstdators personiskās informācijas pierakstīšanai ar speciāla zīmuļa palīdzību, piem., noteiktu datumu un adrešu fiksēšanai, dažādu piezīmju pierakstam. Personālais ciparasistents spēj atpazīt noteiktu rokrakstā rakstītu rakstzīmju kopu.

1197 **Personal Information Manager (PIM)**
**personiskās informācijas pārvaldnieks, *PIM* pārvaldnieks**
**персональный информационный менеджер**

Tekstu apstrādes, datu bāzu un darbvirsmas piederumu programmu apvienojums, kas izveidots personiskās informācijas glabāšanai un uzziņu iegūšanai, piem., par noteiktā laikā un vietā veicamajiem darbiem, personu vārdiem, adresēm, telefona numuriem, maksājumu kārtojumiem.

1198 **Personal NetWare**
**oprētājsistēma *Personal NetWare*, *Personal NetWare***
**операционная система *Personal NetWare***

Firmas *Novell* izstrādāta vienādranga tīkla oprētājsistēma, kas nodrošina oprētājsistēmu *DOS* un *Microsoft Windows* lietotājiem datņu, printeru, atmiņas un citu resursu ērtas koplietošanas iespējas, kā arī vienkāršo tīkla pārvaldību un paaugstina tīkla izmantošanas drošību.

1199 **Pg Dn key**

Sk. *Page Down key.*

1200 **Pg Up key**

Sk. *Page Up key.*

1201 **phototypesetter**
**fotorakstlicis**
**фотонаборный автомат**

Ierīce, kas uz gaismjutīga papīra vai filmas var veidot augstas izšķirtspējas tekstu. Ja ar šīs ierīces palīdzību var izvadīt gan tekstu, gan pustoņu attēlus, tad to parasti sauc par attēllici.

1202 **physical drive**
**fiziskā diskierīce**
**физический дисковод**

Atmiņas ierīce (piem., cietais disks), kura atšķirībā no loģiskās diskierīces reāli eksistē kā atsevišķa ierīce un kurā tiek veiktas informācijas ierakstīšanas un nolasīšanas operācijas. Fizisko diskierīci var nosacīti sadalīt vairākās loģiskajās diskierīcēs.

1203 **PIM**

Sk. *Personal Information Manager.*

1204 **pin compatible**
**kontaktsaderīgs**
**совместимость по контактам**

Īpašība, kas piemīt mikroshēmām vai citiem elektronikas komponentiem ar vienādu kontaktkājiņu skaitu un izvietojumu un kas nodrošina iespēju tās apmainīt ar citām mikroshēmām vai

komponentiem, kuru iekšējās shēmas var būt atšķirīgas, bet izpildāmās funkcijas ekvivalentas.

1205 **pipe**
**programmkanāls**
**программный канал**

Atmiņas daļa, ko viena programma izmanto informācijas izvadei, bet otrai — šī informācija ir ievadinformācija.

1206 **piracy**
**pirātisms**
**пиратизм**

Neatļauta programmatūras vai integrēto shēmu kopēšana personiskai vai komercionālai izmantošanai. Sk. arī *copy protection*.

1207 **pitch**
**[rakstzīmes] iestatne**
**шаг**

Fonta rakstzīmju skaits collā. Šo terminu lieto vienplatuma fontiem, kuru iestatnes vērtība parasti ir 10 vai 12. Proporcionālas platumošanas gadījumā fonta rakstzīmju skaits collā ir mainīgs un iestatni nosaka kā šo rakstzīmju skaita vidējo vērtību.

1208 **pixel (picture element)**
**pikselis**
**элемент изображения, пиксел**

Vismazākais attēla elements, kam rastrgrafikā var tikt patstāvīgi piešķirti tādi raksturojumi kā, piem., krāsa un spilgtums. Sk. arī *bit-mapped graphics*.

1209 **plaintext**
**atklāts teksts**
**открытый текст**

Ziņojums nešifrētā formā. Teksts, kurš nav šifrēts un kuru teksta redaktori un vārdu procesori var brīvi lasīt. Sk. arī *ciphertext, cryptography, decryption, encryption*.

1210 **plasma display**
**plazmas displejs**
**плазменный дисплей**

Plazmas displejs ir viens no elektroluminiscento displeju paveidiem, kurā kā gaismu emitējošā pildījuma slānis izmantota gāze. Šos displejus bieži sauc arī par gāzizlādes displejiem un tos parasti izmanto klēpjdatoros.

1211 **platform**
**platforma**
**платформа**

Datora aparatūras uzbūves pamatprincipi, kas nosaka izmantojamās sistēmas programmatūras veidu, vai arī sistēmas programmatūra, kas nosaka datora lietojumprogrammatūru. Izšķir aparatūras un programmatūras platformas.

1212 **platform independence**
**platformneatkarība**
**платформонезависимость**

1. Programmatūras spēja strādāt ar savstarpēji nesaderīgu aparatūru.
2. Lokālā tīkla spēja strādāt, ja tīklam pievienoti savstarpēji nesaderīgi datori (piem., *IBM PC* saderīgi datori un *Macintosh* saimes datori).

1213 **playback**
**atskaņošana; reproducēšana**
**воспроизведение**

Datu vai signālu nolasīšana no ierakstes vides (magnētiskās lentes vai diska, *CD-ROM* u.c.).

1214 **plot**
**skicēšana**
**набросок изображения**
Attēla veidošana, zīmējot līnijas, kas savieno noteiktus punktus. Skices tiek veidotas, izmantojot šo līniju un punktu koordinātas.

1215 **plotter**
**ploteris**
**графопостроитель**
Ierīce, kas paredzēta datu izvadei no datora grafisku attēlu veidā. Parasti lieto planšetes vai veltņa tipa ploterus. Datiem jābūt vektorgrafikas formā. Attēls tiek veidots no līniju fragmentiem, zīmējot tos uz papīra vai nu ar tinti, vai ar elektrostatisku lādiņu un tonera palīdzību. Ploterus parasti izmanto datorizētas projektēšanas sistēmās un prezentēšanas grafikas sagatavošanai.

1216 **Plug and Play (PnP)**
**standarts *Plug and Play*, standarts *PnP***
**автоматическое конфигурирование аппаратных средств**
Firmu *Compaq*, *Microsoft*, *Intel* un *Phoenix* izstrādāts standarts, ko plaši izmanto aparatūras ražotāji un kas nodrošina automātisku personālo datoru aparatūras konfigurācijas maiņu. Šis standarts ir saderīgs ar operētājsistēmu *Microsoft Windows 95*, kā arī ar sistēmu *BIOS*. Izmantojot šim standartam atbilstošo adapteru atmiņā ierakstīto informāciju, ir iespējams relatīvi vienkārši rekonstruēt datora perifēriju.

1217 **plug compatible**
**spraudņsaderīgs**
**совместимый по разъему**
Īpašība, kas piemīt iekārtām ar vienādu spraudņu konstrukciju un kas ļauj tās savstarpēji apmainīt neatkarīgi no šo iekārtu iekšējās funkcionēšanas īpatnībām (piem., bez savienotājkabeļu pārveidošanas apmainīt dažādu ražotāju izgatavotus modemus vai citas funkcionāli ekvivalentas iekārtas).

1218 **PMS**
Sk. *Pantone Matching System*.

1219 **PnP**
Sk. *Plug and Play*.

1220 **pocket computer**
**kabatdators**
**карманный компьютер**
Portatīvs kalkulatora tipa personālais dators ar autonomu barošanas avotu aktuālas informācijas ierakstīšanai, uzglabāšanai un apstrādei.

1221 **point**
**norādīt; punkts**
**указать; точка**
1. Pārvietot rādītāju pa displeja ekrānu, lai izvēlētos kādu noteiktu objektu.
2. Drukātam izvadam izmantojama tipogrāfiska mērvienība, kas aptuveni vienāda ar 1/72 collas. To parasti izmanto, lai norādītu rakstzīmju augstumu un atstarpes platumu starp teksta rindiņām.

1222 **Point-to-Point Protocol (PPP)**
**divpunktu protokols, protokols *PPP*, *PPP***
**протокол *PPP***
Viens no diviem standartprotokoliem datoru tiešai pievienošanai tīklam *Internet*, izmantojot parasto telefona līniju. Atšķirībā no agrāk lietotā protokola

*SLIP* šis protokols īsteno datu saskaņošanu, saspiešanu un kļūdu labošanu.

**1223 pointer**
**rādītājs**
**указатель**
Simbols, kas attēlots displeja ekrānā bultiņas vai kādā citā formā un kas pārvietojas, pārvietojot peli vai kādu citu rādītājierīci. Norādi izmanto, lai uzrādītu objektus vai darbības, ko lietotājs vēlas atlasīt tālākai izmantošanai. Rādītājs nosaka nevis kursora, bet gan rādītājierīces atrašanās vietu. Sk. arī *click, double-click, mouse pointer.*

**1224 pointing device**
**rādītājierīce**
**указывающее устройство**
Ievadierīce, ko izmanto, lai vadītu kursora kustību vai veidotu grafiskus attēlus displeja ekrānā. Pazīstamākās rādītājierīces ir pele, kursorbumba, kursorsvira, kā arī dažādi gaismas zīmuļi.

**1225 polling**
**aptauja**
**опрос**
Metode, ko izmanto datoru tīklos, lai vadītu pieeju kopējai datu pārraides videi. Datoru tīkla centrālā stacija noteiktā kārtībā aptaujā datoru tīkla perifērijas stacijas, lai noskaidrotu, vai tām ir informācija, ko nepieciešams nosūtīt. Ja šāda informācija ir, tad attiecīgā stacija nosūta centrālajai stacijai apstiprinājumu un uzsāk datu pārraidi.

**1226 POP**
Sk. *Post Office Protocol.*

**1227 pop-down menu**
Sk. *pull-down menu.*

**1228 pop-up menu**
**uznirstošā izvēlne**
**всплывающее меню**
Izvēlne, kas parādās displeja ekrānā, ja lietotājs ar rādītājierīces starpniecību izvēlas kādu objektu, piem., tekstu, rītjoslu vai dialoglodziņu. Ja lietotājs izvēlas noteiktu izvēlnes elementu, tad šī izvēlne no displeja ekrāna pazūd.

**1229 pop-up window**
**uznirstošs logs**
**всплывающее окно**
Kustīgs fiksēta lieluma logs, kurā lietotājs ievieto informāciju, ko pieprasa lietojumprogramma un kas nodrošina tās izpildes procesa turpināšanu atbilstoši lietotāja prasībām.

**1230 port**
**pieslēgvieta, ports**
**порт**
Fizikāls savienojums, ar kura starpniecību sinhronizē un vada datu plūsmu starp centrālo procesoru un tādām ārējām iekārtām kā, piem., modemiem un printeriem. Sk. arī *parallel port, serial port.*

**1231 port expander**
**pieslēgvietas izvērsējs**
**расширитель порта**
Aparatūra, ar kuru vairākas sakaru līnijas (iekārtas) pievieno vienai pieslēgvietai. Sakaru līnijas pieeju pieslēgvietai nodrošina ar speciālas programmatūras vai slēdža palīdzību.

**1232 portability**
**pārnesamība**
**переносимость**
Programmas spēja tikt izpildītai dažāda tipa datu apstrādes sistēmās, nepārveidojot to citā programmēšanas valodā un neveicot nozīmīgas tās modifikācijas.

1233 **portable computer**
**portatīvs dators**
**портативный компьютер**
Personālais dators, kura konstrukcija un svars ļauj to viegli pārvietot. Sk. *laptop computer, notebook computer, palmtop computer, pocket computer.*

1234 **portrait orientation**
**portretorientācija**
**портретноориентированность**
Izdrukājamais dokuments ir portretorientēts, ja tā augstums ir lielāks par tā platumu.

1235 **Post Office Protokol (POP)**
**pasta nodaļas protokols, protokols *POP*, *POP***
**протокол почтового отделения связи**
Standartprotokols, ko izmanto elektroniskā pasta lietotāji, lai saņemtu ziņojumus no elektroniskā pasta servera.

1236 **posting**
**apziņošana**
**уведомление**
Elektroniskā pasta ziņojuma nosūtīšana kādai datoru tīkla interešķopai vai adresātu sarakstam, nevis individuālam adresātam. Parasti šo terminu izmanto tīkla *USENET* lietotāji.

1237 **postmaster**
**pasta pārzinis**
**почтмейстер**
Datoru tīklā — persona, kas veic elektroniskā pasta pārvaldību un risina ar tā darbību saistītas problēmas. Šo terminu lieto arī kā aizstājvārdu elektroniskā pasta serverim, kurš veic maršrutēšanas vadību.

1238 **postprocessor**
**pēcprocesors**
**постпроцессор**
1. Datora programma, ar kuru veic datu apstrādes noslēguma operācijas, piem., datu formatēšanu, lai tos parādītu displeja ekrānā vai izdrukātu.
2. Procesors, kas veic pamatprocesora darbības gaitā iegūto rezultātu papildapstrādi. Sk. arī *preprocesor.*

1239 **PostScript**
**valoda *PostScript*, *PostScript***
**язык *PostScript***
Lappušu aprakstvaloda kvalitatīvai tekstu un grafikas drukāšanai, izmantojot lāzerprinterus vai citas drukas iekārtas ar augstu izšķirtspēju. To parasti lieto datorizdevniecībā. Sk. arī *PostScript printer.*

1240 **PostScript font**
***PostScript* fonts**
**шрифт языка *PostScript***
*PostScript* lappušu apraksta valodā definēts fonts, kurš var tikt izdrukāts ar *PostScript* printeri. *PostScript* fonti atšķiras no bitkartētiem fontiem ar savu nogludinātību un augsto kvalitāti. Vairumā *PostScript* lāzerprinteru biežāk lietojamie *PostScript* fonti jau ir iebūvēti un to ielāde nav nepieciešama.

1241 **PostScript printer**
***PostScript* printeris**
**печатающее устройство *PostScript***
Printeris (parasti lāzerprinteris), kurā ir speciāls mikroprocesors un 1 megabaits brīvpieejas atmiņas, kas nepieciešama, lai valodā *PostScript* formulētās instrukcijas dekodētu un interpretētu drukāšanai. Sk. arī *PostScript font.*

1242 **power supply**
**barošanas avots**
**источник питания**
Elektriska ierīce, kas apgādā datoru ar elektroenerģiju. Pieslēdzot barošanas avotu elektriskajam tīklam, tajā parasti notiek maiņstrāvas pārvēršana vajadzīgā sprieguma līdzstrāvā, sprieguma stabilizācija un pulsāciju filtrēšana.

1243 **power user**
**prasmīgs lietotājs**
**квалифицированный пользователь**
Persona, kas prot profesionāli strādāt ar datoru un kas izmanto lietojumprogrammu paplašinātas iespējas, piem., izklājlapas vai teksta procesorus.

1244 **PowerPC**
**arhitektūra *PowerPC***
**архитектура *PowerPC***
Atvērta personālo datoru arhitektūra, ko 1992. g. izveidoja firmas *IBM, Motorola* un *Apple*. Arhitektūra *PowerPC* izstrādāta uz *RISC* datoru un superskalāro procesoru arhitektūras bāzes. Šīs arhitektūras mikroprocesoriem ir 64 bitu datu kopnes un 32 bitu adrešu kopnes, datu un instrukciju kešatmiņas, kā arī fiksētā un peldošā komata procesori. Vairāki šīs arhitektūras mikroprocesori, kas galvenokārt atšķiras ar ātrdarbību un vienā taktī izpildāmo instrukciju skaitu, tiek ražoti vai arī atrodas dažādās izstrādes stadijās (*MPC 601, MPC 602, MPC 603, MPC 604, Power 2* u.c.).

1245 **Precision Architecture (PA)**
**arhitektūra *PA***
**архитектура *PA***
Firmas *Hewlett-Packard* 80. gados izstrādāta *RISC* datora arhitektūra vienotai mikrodatoru un minidatoru saimei. Šī arhitektūra tika izveidota, lai apmierinātu prasības, kas izvirzītas visiem firmas *Hewlett-Packard* izstrādājumiem ar 32 bitu adrešu un datu formātiem. Arhitektūra *PA* paredz instrukciju un datu kešatmiņu, kā arī trīs veidu palīgprocesoru lietošanu. Izmantojot šo arhitektūru, izveidots, piem., ātrdarbīgs centrālais procesors *PA 7100*, kura darba frekvence ir 100 MHz.

1246 **preprocessor**
**priekšprocesors**
**препроцессор**
Datora programma vai procesors, kas veic pirmprogrammas vai datu iepriekšēju apstrādi pirms to nodošanas tālākai apstrādei pamatprogrammai vai pamatprocesoram. Sk. arī *postprocessor*.

1247 **presentation graphics**
**priekšstādes grafika,**
**prezentēšanas grafika**
**представительная графика**
Uzskatāms komercinformācijas grafisks attēlojums, piem., joslu diagrammu vai grafiku veidā, ko parasti izmanto, lai vizuāli attēlotu darījumu vai finansiālā stāvokļa raksturojošo skaitlisko informāciju un tās izmaiņas.

1248 **preventive maintenance**
**preventīvā uzturēšana**
**профилактика**
Uzturēšana, kas novērš sistēmas funkcionēšanas traucējumu rašanās iespēju.

1249 **preview**
**priekšskatījums**
**предварительный просмотр**
Grafikas vai teksta lappuses parādīšana displeja ekrānā tādā izpildījumā, kādā tā tiks izdrukāta.

1250 **primary cache**
**primārā kešatmiņa**
**первичная кэш-память**
Kešatmiņa, kas atšķirībā no sekundārās kešatmiņas ir iebūvēta mikroprocesorā, nevis izvietota uz mātesplates.

1251 **primary key**
**primārā atslēga**
**первичный ключ**
Relāciju datu bāzes pārvaldības sistēmā — unikāls lauks, kas identificē datu bāzes tabulas rindiņu. Šo atslēgu dažkārt sauc arī par galveno atslēgu.

1252 **primary storage**
**primārā atmiņa**
**основная память**
Datora operatīvā atmiņa, kas sastāv no brīvpieejas atmiņas un lasāmatmiņas, kurām ir tieša pieeja no centrālā procesora. Sk. arī *secondary storage*.

1253 **primary window**
**primārais logs**
**первичное окно**
Logs, kurā notiek galvenā mijiedarbība starp lietotāju un lietojumprogrammu. Sk. arī *pop-up window, secondary window*.

1254 **Print Manager**
**lietojumprogramma *Print Manager***
**прикладная программа *Print Manager***
Lietojumprogramma, kas paredzēta lietošanai operētājsistēmas *Microsoft Windows* vidē un kas koordinē visu operētājsistēmas *Microsoft Windows* lietojumprogrammu izpildes rezultātu drukāšanu.

1255 **print modifiers**
**drukas modifikatori**
**модификаторы печати**
Kodi dokumentā, kas liek printerim mainīt burtstilu, piem., pāriet no treknraksta uz slīprakstu.

1256 **print queue**
**drukdarbu rinda**
**очередь печати**
Datņu saraksts, kas ar drukas spolētāja starpniecību tiek izdrukāts kā fona darbs tajā laikā, kad dators izpilda citus, priekšplāna darbus.

1257 **Print Screen key**
**ekrāndrukāšanas taustiņš**
**клавиша распечатки экрана**
Firmas *IBM* un ar to saderīgas tastatūras taustiņš, kas parasti izvietots tastatūras augšpusē un tiek izmantots, lai ekrāna saturu nosūtītu printerim drukāšanai. Dažās programmās ekrāndrukāšanas taustiņu izmanto, lai ekrāna saturu varētu ierakstīt datnē vēlākai tā apstrādei.

1258 **print server**
**drukas serveris**
**сервер печати**
Serveris, kas parasti vada vairākus printerus. Drukas serveris no citiem tīklā apvienotajiem datoriem saņem un uzkrāj drukājamās datnes, veido to rindu un vienu pēc otras nodod tās drukāšanai vienam vai vairākiem tam pievienotajiem printeriem.

1259 **print spooler**
**drukas spolētājs**
**блок перекачки данных для печати**
Palīgprogramma, kas nodrošina īslaicīgu

drukājamo datņu uzglabāšanu drukas rindā un nodod tās vienu pēc otras printerim drukāšanai. Sk. arī *print server*.

**1260 printed circuit**
**drukātā shēma**
**печатная схема**

Izolācijas materiāla plate, uz kuras ar speciālas tehnoloģijas palīdzību veido elektriskos savienojumus. Drukātās shēmas izgatavošanai parasti izmanto stikltekstolita plati, kas pārklāta ar vara foliju. Pēc shēmas uzdrukāšanas liekos folijas laukumus izkodina. Visbiežāk lieto vienpusējās vai divpusējās drukātās shēmas.

**1261 printed circuit board**
**drukātās shēmas plate**
**плата с печатным монтажом**

Drukātā shēma, uz kuras uzmontēti mehāniskie, elektriskie un elektroniskie komponenti, kas kopā ar esošajiem savienojumiem veido konstruktīvi viegli nomaināmu sistēmas funkcionālo bloku.

**1262 printer**
**printeris**
**принтер**

Izvadierīce, kas saņem no datora kodētus datus, pārveido tos lasīšanai piemērotā formā un izveido atbilstošu tekstu vai grafiku uz papīra vai citā datu vidē. Sk. arī *color printer, dot matrix printer, electrostatic printer, ink jet printer, laser printer, light emitting diod printer, liquid crystal printer, matrix printer, parallel printer, serial printer, thermal transfer printer, thermal vax-transfer printer.*

**1263 printer font**
**printera fonts**
**шрифт принтера**

Fonts, kas pastāvīgi atrodas printera atmiņā vai tiek uz to pārsūtīts. Printera fonts var būt iebūvēts fonts, ielādējams fonts vai arī fontu kasetnes fonts.

**1264 privacy**
**privātums**
**конфиденциальность**

Fizisku vai juridisku personu tiesības pārraudzīt datu krājumus un izmantot visus tajos esošos datus vai arī tikai tos datus, kas attiecas uz šīm personām.

**1265 Privacy Enhanced Mail (PEM)**
**paaugstināta privātuma pasts, standarts *PEM***
**почта повышенной секретности**

Tīkla *Internet* standarts, kas elektroniskā pasta pakalpojumiem nodrošina paaugstinātu privātuma pakāpi. Izmantojot standartu *PEM*, ar speciālas šifrēšanas un atšifrēšanas starpniecību nodrošina, ka elektroniskā pasta ziņojumu var izlasīt vienīgi tā tiešais adresāts.

**1266 privacy lock**
**privātuma slēdzene**
**замок секретности**

Datu bāzes pārvaldības sistēmai zināms kods, kas piekārtots aizsargājamajam resursam un nodrošina tā slepenību.

**1267 problem-oriented language**
**problēmorientēta valoda**
**проблемно-ориентированный язык**

Programmēšanas valoda, kas izstrādāta kādas noteiktas uzdevumu klases risināšanai, piem., sarakstu apstrādes valoda, informācijas izguves valoda, modelēšanas valoda.

1268 **procedure-oriented language**
**procedūrorientēta valoda**
**процедурно-ориентированный язык**

Programmēšanas valoda, kurā lietotājs izveido īpašu instrukciju kopu, kas datoram noteiktā secībā jāizpilda. Pie šī tipa valodām pieder vairums plaši izmantojamo programmēšanas valodu.

1269 **processing**
**apstrāde**
**обработка**

Programmā paredzēto instrukciju izpilde datora centrālajā procesorā, lai pēc noteiktiem kritērijiem pārveidotu, kārtotu un atlasītu datus vai izdarītu ar tiem matemātiskus aprēķinus. Sk. arī *cooperative processing, data processing, image processing, interactive processing, word processing.*

1270 **processor**
**procesors**
**процессор**

Sk. *central processing unit; microprocessor.*

1271 **professional workstation**
**profesionālā darbstacija**
**профессиональная рабочая станция**

Augstas veiktspējas personālais dators, kas piemērots izmantošanai dažādiem profesionāliem lietojumiem tādās jomās kā diskrēto shēmu projektēšana, arhitektūra u.c. Profesionālajās darbstacijās parasti izmanto augstas izšķirtspējas displejus, ātrdarbīgus un jaudīgus mikroprocesorus, lielu atmiņas apjomu un operētājsistēmu *UNIX*.

1272 **program**
**programma**
**программа**

Instrukciju kopa, kas nosaka operāciju secību, ko izpilda dators datu apstrādes procesā. Programma tiek rakstīta kādā no programmēšanas valodām. Programmas parasti tiek iedalītas sistēmprogrammās, palīgprogrammās un lietojumprogrammās.

1273 **program file**
**programmu datne**
**файл программ**

Datne, kurā ir programmu izpildāmās daļas, piem., vārdu procesors, izklājlapu programma, komunikāciju programmu pakotnes, dažādas lietojumprogrammas.

1274 **program generator**
**programmu ģenerators**
**генератор программ**

Programmatūra, kas ļauj lietotājam uzrakstīt sarežģītas programmas, izmantojot nelielu skaitu relatīvi vienkāršu instrukciju.

1275 **Program Manager**
**čaula *Program Manager*,**
***Program Manager***
**оболочка *Program Manager***

Operētājsistēmas *Microsoft Windows* čaulas programma, kas īsteno lietojumprogrammu un uzdevumu izpildes pārvaldību.

1276 **programmable calculator**
**programmējams kalkulators**
**программируемый калькулятор**

Skaitlotājs, kas var apstrādāt tikai skaitliskus, bet ne burtciparu datus.

1277 **programmable function key**
**programmējams funkcijtaustiņš**
**программируемая функциональная клавиша**

Tastatūras taustiņš, kas var izpildīt dažādas funkcijas, pārtverot tastatūras kodus un aizstājot tos ar iepriekš sagatavotām taustiņu kombinācijām vai taustiņsitienu sekvencēm (makrokomandām).

1278 **Programmable Read-Only Memory (PROM)**
**programmējamā lasāmatmiņa**
**программируемое постоянное запоминающее устройство**

Pusvadītāja lasāmatmiņa, no kuras datus var tikai lasīt. Programmējamā lasāmatmiņā datus ieraksta vai nu tās izgatavošanas procesā vai arī to veic lietotājs ar speciālas ierīces — programmatora — palīdzību. Atšķirībā no pārprogrammējamās lasāmatmiņas šī tipa atmiņas mikroshēmas var programmēt tikai vienu reizi. Sk. arī *Erasable Programmable Read-Only Memory, Read Only-Memory.*

1279 **Programmer's Hierarchical Interactive Graphics Standard (PHIGS)**
**standarts *PHIGS*, *PHIGS***
**стандарт *PHIGS***

Standarts, kas nodrošina aparatūrneatkarīgu saskarni starp lietojumprogrammatūru un grafikas adapteri un kas izmanto standartkomandu kopu, lai zīmētu vai pārveidotu divu un trīs dimensiju attēlus. Sk. arī *Graphics Kernel System.*

1280 **programming**
**programmēšana**
**программирование**

Programmu izstrādāšana datoram. Programmēšana ietver problēmas risināšanas algoritma detalizēšanu un tā pierakstīšanu attiecīgajā programmēšanas valodā, datu struktūras izvēli un to kodēšanu, kā arī izveidotās programmas atkļūdošanu. Plašākā nozīmē ar šo terminu apzīmē arī programmēšanas teoriju.

1281 **PROM**

Sk. *Programmable Read-Only Memory.*

1282 **prompt**
**uzvedne**
**подсказка**

Displeja ekrānā izspīdināts teksts, kas norāda, ka lietotājam jāievada zināma informācija.

1283 **proportional font**
**proporcionālais fonts**
**пропорциональный шрифт**

Noteiktu izmēru rakstzīmju kopa, kurā katra rakstzīme aizņem dažāda platuma horizontālu telpu, piem., burts "i" aizņem mazāku telpu nekā burts "m". Sk. arī *fixed pitch, monospace font, proportional pitch.*

1284 **proportional pitch**
**proporcionālā iestatne**
**пропорциональное размещение**

Rakstzīmes faktiskajam platumam proporcionālas horizontālas vietas izmantošana drukātos tekstos. Sk. arī *fixed pitch, monospace font, monospacing, proportional font, proportional spacing.*

1285 **proportional spacing**
**proporcionālā platumošana**
**соразмерное распределение пробелов**
Telpas iedalīšana rakstzīmju drukāšanai vai attēlošanai displeja ekrānā proporcionāli rakstzīmes platumam. Sk. arī *kerning, monospacing*.

1286 **protected mode**
**aizsargrežīms**
**защищенный режим**
Firmas *Intel* mikroprocesoru (sākot ar *Intel 80286*) darbības režīms, kas nodrošina virtuālo atmiņu un vairākuzdevumu režīmu, dodot iespēju vienlaicīgi izpildīt vairākas programmas.

1287 **protocol suite**
**protokolu komplekts**
**комплект протоколов**
Savstarpēji saistītu protokolu kopums, kas nosaka datoru tīkla arhitektūru un darbību. Tā, piem., tīkls *Internet* izveidots uz protokolu *TCP/IP* komplekta bāzes un šie protokoli izstrādāti tā, lai to darbība būtu saskanīga.

1288 **PS/2**
Sk. *IBM PS/2*.

1289 **PS/2 bus**
**kopne PS/2**
**шина PS/2**
Sk. *Micro Channel Architecture*.

1290 **public domain software**
**publiskā programmatūra**
**общедоступное программное обеспечение**
Programmatūra, ko neaizsargā autortiesības un ko var izplatīt bez maksas, neprasot tās autora atļauju.

1291 **pull-down menu**
**izvelkamā izvēlne**
**спускающееся меню**
Izvēlne, kuras vārds ir izvēlņu joslā un kura parādās displeja ekrānā izvērstā veidā, ja lietotājs to izvēlējies, izmantojot peli. Izvelkamā izvēlne ir redzama displeja ekrānā tikmēr, kamēr peles poga ir nospiesta. Lai izdarītu izvēli, ar peli pārvieto kursoru uz vajadzīgo izvēlnes elementu un atlaiž peles pogu.

1292 **push button**
**spiežampoga**
**кнопка запуска, нажимаемая кнопка**
Grafiskajā lietotāju saskarnē — lielāka izmēra poga, kas dialoglodziņā iniciē lietotāja izvēlētās darbības. Vairumā dialoglodziņu ir, piem., apstiprinājumpoga, kas apstiprina lietotāja izvēli un izpilda komandu, kā arī atcelšanas poga, kas atceļ lietotāja izvēli un aizver dialoglodziņu. Dialoglodziņā parasti ir arī noklusējuma poga, kas pārstāv biežāk izmantojamās lietotāja izvēles. Sk. arī *command button*.

## Q

1293 **quadrature amplitude modulation**
**amplitūdas kvadratūrmodelēšana**
**квадратурная амплитудная модуляция**
Datu kodēšanas metode, kuru izmanto ātrdarbīgos modemos un kurā apvienota amplitūdas un fāzes modulēšana, lai palielinātu datu pārraides ātrumu.

1294 **quality assurance**
**kvalitātes nodrošināšana**
**обеспечение качества**
1. Sistematizēta visu to darbību shēma, kas nepieciešamas, lai nodrošinātu izstrādnes atbilstību konkrētām tehniskām prasībām.
2. Darbību kopums, kas speciāli radīts, lai novērtētu produkta izstrādāšanas un ražošanas procesu.

1295 **Quattro Pro**
**izklājlapu programma *Quattro Pro*, *Quattro Pro***
**программа табличных вычислений *Quattro Pro***
Firmas *Borland International Inc.* izstrādāta izklājlapu programma ar visai plašām iespējām. Tā izmantojama kā operētājsistēmas *DOS*, tā arī operētājsistēmas *Microsoft Windows* vidē. Sk. arī *Microsoft Excel, Lotus 1-2-3.*

1296 **query**
**vaicājums**
**запрос**
1. Lietotāja vēršanās pie datu bāzes, lai iegūtu nepieciešamo informāciju.
2. Interaktīvās sistēmas lietotāja darbība, kas pieprasa sistēmas atbildi.

1297 **query language**
**vaicājumvaloda**
**язык запросов**
Lietotāja un informācijas sistēmas mijiedarbības valoda, kuru izmanto, lai aprakstītu informācijas meklēšanas un izvades vaicājumu.

1298 **query-by-example (QBE)**
**piemērvaicājums, vaicājums *QBE***
**запрос по образцу**
Process, kurā izgūst informāciju no datu bāzes. Meklēšanas nosacījumi tiek ievadīti displeja ekrānā attēlotā veidnē, kur redzamas datu bāzes rindiņas un kolonnas. Izmantojot piemērvaicājumu, lietotājs tiek atbrīvots no vaicājumvalodas apgūšanas.

1299 **question mark**
**jautājuma zīme**
**вопросительный знак**
Aizstājzīme, ko operētājsistēmās *MS-DOS, Windows NT* un *OS/2* izmanto kādas noteiktas rakstzīmes vietā.

1300 **QWERTY**
***QWERTY* tastatūra**
**клавиатура *QWERTY***
Datoru (arī rakstāmmašīnu) angļu valodas tastatūra, kuras augšējās burtu taustiņu rindas pirmie seši burti ir *Q, W, E, R, T, Y.*

## R

1301 **ragged margin**
**robaina mala**
**неровный край [текста]**
Dokumenta mala, kurā rindiņas nesākas vienā un tajā pašā pozīcijā (kreisā robainā mala) vai nebeidzas vienā un tajā pašā pozīcijā (labā robainā mala).

1302 **RAM**
Sk. *Random-Access Memory.*

1303 **RAM disk**
**brīvpieejas atmiņas disks, *RAM* disks**
***диск произвольного доступа***
Atmiņas iekārtā emulēts diskdzinis. Lai

izmantotu brīvpieejas atmiņas disku, tajā vispirms no magnētiskā diska pārkopē programmas un datnes. Ievadizvades operācijas, kas parasti saistītas ar magnētisko disku, tiek novirzītas uz *RAM* disku, kas ievērojami paātrina informācijas apstrādi. Iekārtai atslēdzoties, zūd *RAM* diskā ierakstītā informācija. Lai tas nenotiktu, informācija savlaicīgi jāpārkopē fiziskajā diskā.

1304 **random access**
**brīvpieeja**
**произвольный доступ**

Informācijas uzglabāšanas un izguves metode. Atšķirībā no secīgās pieejas brīvpieeja ļauj tieši vērsties pie vajadzīgajiem datiem, neatkarīgi no to atrašanās vietas datora atmiņā. Izmantojot šo pieejas metodi, ievērojami samazinās vajadzīgo datu atrašanas laiks, jo nav nepieciešams caurskatīt pirms tiem ierakstītos datus.

1305 **Random-Access Memory (RAM)**
**brīvpieejas atmiņa, *RAM* atmiņa**
**запоминающее устройство с произвольной выборкой (ЗУПВ)**

Datora primārās (operatīvās) atmiņas daļa, kurā uzglabātajām programmu instrukcijām un datiem ir iespējama tieša pieeja no centrālā procesora, izmantojot ātrdarbīgo ārējo kopni. Atšķirībā no otra primārās atmiņas komponenta — lasāmatmiņas — no brīvpieejas atmiņas centrālais procesors var ne tikai nolasīt datus, bet var datus tajā arī ierakstīt. Sk. arī *dynamic random-access memory, static random access memory.*

1306 **range**
**diapazons**
**диапазон**

Izklājlapu programmās — viena šūna vai blakusesošu šūnu taisnstūrveida grupa. Šūnu grupējums ļauj izdarīt zināmas operācijas, piem., formatēšanu, visai šūnu grupai.

1307 **raster**
**rastrs**
**растр**

Horizontālu līniju šablons, ko izmanto attēlu formēšanai displeja ekrānā. Katra rastra līnija sastāv no punktiem, ko sauc par pikseļiem. Pikseļus var izgaismot atsevišķi, tādējādi veidojot rakstzīmes un grafiku.

1308 **raster display**
**rastra displejs**
**растровый дисплей**

Displejs, kas izspīdina attēlus ekrānā, izmantojot horizontālas skenēšanas līnijas. Rastra displejos attēlu veido, izgaismojot atsevišķus pikseļus uz katras no šīm līnijām.

1309 **raster graphics**
**rastrgrafika**
**растровая графика**

Grafiku vai zīmējumu attēlošanas veids, kurā līnijas displeja ekrānā veidotas no atsevišķiem punktiem (rastra elementiem). Katra punkta krāsu un gaišumu nosaka programma. Sk. arī *vector graphics.*

1310 **raster image processor (RIP)**
**rastrattēla procesors, *RIP* procesors**
**процессор растрового изображения**

Procesors un programmatūra, kas pārveido vektorgrafiku un tekstu rastrgrafikas (bitkartētā) attēlā vai pilnīgi noformētā lappusē, kuru pēc tam izdrukā, izmantojot printerus, elektrostatiskos ploterus vai attēlličus.

1311 **raster-to-vector conversion**
**rastrvektora konversija**
**растро-векторное преобразование**

Bitkartētu attēlu pārveidošana vektoros vai līniju segmentos. Sk. arī *bit mapped graphics, vector-to-raster conversion.*

1312 **Read-Only Memory (ROM)**
**lasāmatmiņa, *ROM* atmiņa**
**постоянное запоминающее устройство (ПЗУ)**

Energoneatkarīga datora primārās (operatīvās) atmiņas daļa, kuras saturu ieraksta tās izgatavotājs un kuru datora darbības gaitā nevar mainīt. Lasāmatmiņā parasti tiek ierakstītas būtiskas sistēmprogrammas un lasāmatmiņu izmanto arvien lielāku operētājsistēmas daļu uzglabāšanai. Sk. arī *Erasable Programmable Read-Only Memory, Programmable Read-Only Memory, Random-Access Memory.*

1313 **read/write head**
**lasīšanas/rakstīšanas galviņa**
**головка чтения/записи**

Ierīce, kas, pārvietojoties pa diskatmiņas virsmu, veic datu ierakstīšanu un nolasīšanu.

1314 **README file**
**datne LASIMANI, datne *README***
**файл *README***

Teksta datne, kuru bieži ietver lietojumprogrammu instalācijprogrammās, un kurā ir jaunākā informācija, kas vēl nav atspoguļota programmas dokumentācijā.

1315 **real mode**
**reālrežīms**
**реальный режим**

Firmas *Intel* mikroprocesoru darba režīms, kurā katrai programmai atmiņā ir noteikts izvietojums un nodrošināta tieša pieeja atmiņai un ievadizvades ierīcēm. Atmiņā vienlaicīgi nevar tikt apstrādātas vairākas programmas.

1316 **real-time**
**reālais laiks, reāllaiks**
**реальное время**

Faktiskais datu apstrādes procesa norises laiks, kas atkarīgs no datu plūsmas, ko dators saņem no kāda ārēja procesa. Apstrādes laika ierobežojumus nosaka ārējā procesa raksturs. Terminu lieto arī, aprakstot dialogrežīmu, kā arī citus procesus, kuru norises gaitā iespējama cilvēka iejaukšanās.

1317 **real-time chat**
**reāllaika tērzēšana**
**разговор в реальном времени**

Viens no tīkla *Internet* lietojumiem, kas, izmantojot datora tastatūru, dod iespēju īstenot sarunu dialoga režīmā.

1318 **reboot**
**atsāknēt**
**повторять начальную загрузку**

Uzsākt darbināt datoru, atkārtoti ielādējot operētājsistēmu.

1319 **record**
**ieraksts**
**запись**

1. Saistītu datu elementu kopums, ko datorā apstrādā kā vienotu veselumu. Saistītu ierakstu kopums veido datni.
2. Datu bāzes struktūras elements, kas tiek uzglabāts datu laukos, kuriem piešķirts noteikts vārds. Sk. arī *row*.

1320 **record locking**
**ierakstu bloķēšana**
**блокировка записи**

Metode, ko izmanto dalītās datu apstrādes vai citās vairāklietotāju sistēmās un kas vienlaicīgi tikai vienam lietotājam ļauj izdarīt izmaiņas ierakstā.

1321 **record-oriented database management program**
**ierakstorientēta datu bāzes pārvaldības programma**
**программа управления базой данных, ориентированная на работу с записями**

Datu bāzes pārvaldības programma, kas atšķirībā no tabulorientētām datu bāzes pārvaldības programmām, vaicājumoperācijas rezultātā displeja ekrānā izspīdina ierakstus, nevis tabulas. Sk. arī *retrieval*, *database management system*, *Structured Query Language*.

1322 **Recordable Compact Disk (CD-R)**
**ierakstāms kompaktdisks, kompaktdisks *CD-R*, *CD-R***
**компактдиск с записью**

Kompaktdisks, no kura informāciju var ne tikai operatīvi nolasīt, bet kurā to var arī ierakstīt. Informācijas nolasīšanai no ierakstāmā kompaktdiska var izmantot jebkuru standarta *CD-ROM* atskaņotāju, bet informācijas ierakstīšanai nepieciešams speciāls *CD-R* diskdzinis.

1323 **recovery**
**atkopšana**
**восстановление**

Saglabājamās informācijas atjaunošana pēc datora kļūmes novēršanas. Šaurākā nozīmē — datu bāzes rekonstrukcija, izmantojot, piem., dublētājkopijas vai modificētu ierakstu bloku kopijas.

1324 **recycle bin**
**atkritne**
**бункер повторного использования**

Ikona, kas, strādājot ar operētājsistēmu *Microsoft Windows 95*, displeja ekrānā norāda, kur tiek uzglabātas atmestās datnes. Atkritnē uzglabātās datnes var tikt atjaunotas vai arī pilnīgi likvidētas.

1325 **redirection**
**virzienmaiņa**
**переназначение**

Datu novirzīšana no to parastās saņēmējierīces uz citu, piem., direktorija saraksta pārsūtīšana no displeja ekrāna uz printeri nevis tā ierakstīšana diskatmiņā. Sk. arī *destination*, *source*.

1326 **Reduced Instruction Set Computer (RISC)**
**sašaurinātās instrukcijkopas dators, *RISC* dators**
**компьютер с сокращенным набором инструкций**

Dators (procesors) ar relatīvi mazu instrukciju kopu. Šī tipa datoru instrukcijām ir vienāds formāts un katra no tām parasti tiek izpildīta vienā taktī, kas nodrošina ātrdarbības palielināšanu. Sk. arī *Complex Instruction Set Computer*.

1327 **refresh rate**
**atsvaidzes intensitāte**
**интенсивность регенерации**
Attēla atkārtotas zīmēšanas reižu skaits sekundē katodstaru lampu displejos.

1328 **refreshing**
**atsvaidzināšana**
**регенерация**
1. Datu saglabāšana datora atmiņā, tos periodiski atjaunojot. Sk. arī *Dynamic Random-Access Memory (DRAM)*.
2. Periodiska informācijas attēlojuma atkārtošana displeja ekrānā.

1329 **rehyphenation**
**zilbjpārdale**
**переразбивка текста по слогам**
Vārdu zilbjdales pārveidojums, kas nepieciešams pēc teksta lappušu formāta vai rindiņas platuma maiņas.

1330 **relation**
**attiecība; relācija**
**отношение**
1. Vairāku objektu saistības izpausme, kas attēlo tiem kopīgi piemītošo.
2. Relāciju datu bāzu modeļos — tabula, ko izmanto kā informācijas uzdošanas pamatformu.

1331 **relational database**
**relāciju datu bāze**
**реляционная база данных**
Datu bāze, kurā viena tipa ierakstos ir norādes uz cita tipa ierakstiem. Relāciju datu bāze dod iespēju lietotājam saistīt informāciju, kas tiek glabāta dažādās datnēs, kā arī veidot noteiktas attiecības starp dažāda tipa ierakstiem.

1332 **relational model**
**relāciju modelis**
**реляционная модель**
Datu modelis, kurā dati organizēti, izmantojot savstarpējās attiecības, kas parasti atspoguļotas tabulas formā. Šis datu modelis tiek izmantots vairumā mūsdienu datu bāzu pārvaldības sistēmās. Sk. arī *relational database*.

1333 **reliability**
**drošums; uzticamība**
**надежность**
Datora aparatūras vai programmatūras spēja izpildīt lietotāja prasības bez kļūmēm vai funkcionēšanas traucējumiem.

1334 **removable disk**
**noņemams disks**
**сменный диск**
Sk. *disk cartridge*, *diskette*.

1335 **rendering**
**renderēšana**
**рендеринг**
Datorgrafikā — kontūrzīmējuma pārveidošana pilnībā noformētā trīsdimensiju attēlā, izmantojot krāsas un ēnojumus.

1336 **repagination**
**lappārdale**
**перераспределение страниц**
Tekstapstrādes un datorizdevniecības operācija, kas pārnumurē lappuses gadījumos, kad tiek iesprausts vai izmests teksts, pārvietotas rindkopas vai arī izdarītas kādas citas izmaiņas.

1337 **report**
**pārskats**
**отчет**

Informācija par kādu noteiktu jautājumu, ko datu apstrādes sistēma sagatavo lietotājam. Pārskats parasti tiek attiecīgi formatēts un sagatavots drukātā veidā. Tā, piem., datu bāzu programmatūra nodrošina pārskatu sastādīšanu, kuros ir lappušu numuri, galvenes, ievadītā un izskaitļotā informācija. Citos lietojumos pārskats var ietvert tekstu, grafiku un diagrammas.

1338 **Request For Comments (RFC)**
***RFC* dokumenti**
**запрос на комментарий**
Dokumenti, ko izplata tīklā *Internet*. Dokumentos aprakstīti tīkla protokoli (t.sk. visi tīkla *Internet* standarti), kā arī to pārbaudes un izmantošanas rezultāti. Atšķirībā no citām protokolu sistēmām, kuras parasti apstiprina dažādas oficiālas standartizēšanas organizācijas, tīkla *Internet* protokolu projektus izplata to izstrādātāji, tie ir brīvi pieejami tīkla lietotājiem un ar laiku (pēc komentāru apkopošanas) tie var kļūt par faktiskajiem standartiem Sk. arī *For Your Information*.

1339 **reset**
**atgriešana**
**возврат; сброс**
Sistēmas (iekārtas, ierīces) iestādīšana sākuma vai kādā citā iepriekš noteiktā stāvoklī. Šī procedūra parasti saistīta ar bojājumu vai kļūdu atklāšanu sistēmas (iekārtas, ierīces) darbībā, vai ar tās darbības atsākšanu pēc pārtraukuma.

1340 **reset button**
**atgriešanas poga**
**кнопка перезагрузки**
Poga vai slēdzis, kas ļauj izpildīt datora restartu — veikt silto sāknēšanu.

1341 **resident font**
**rezidents fonts**
**резидентный фонт**
Sk. *internal font*, *downloadable font*, *printer font*.

1342 **resolution**
**izšķirtspēja**
**разрешающая способность**
Attēla kvalitātes novērtējums, ko izmanto, lai salīdzinātu dažādus videostandartus un printerus. Videostandartos izšķirtspēja nozīmē punktu (pikseļu) skaitu displeja ekrānā. Piemēram, 640×480 pikseļu ekrāns attēla veidošanai ir spējīgs izmantot 640 atsevišķus punktus katrā no 480 rindām vai apmēram 3 miljonus pikseļu. Punktmatricas vai lāzera printeros izšķirtspēja nosaka punktu skaitu collā. Piemēram, printeris ar izšķirtspēju 300 punkti collā ir spējīgs drukāt 300 atsevišķus punktus rindā, kuras garums ir 1 colla, vai arī 90000 punktu vienā kvadrātcollā.

1343 **resource**
**resursi**
**ресурс**
Jebkurš datoru sistēmas komponents, kas tiek izmantots programmu izpildes procesā, piem., operatīvā atmiņa, diskatmiņa, ievadizvades ierīces. Dažkārt par resursiem tiek uzskatīti arī dialoglodziņi, bitkartes un fonti.

1344 **resource allocation**
**resursu iedalīšana**
**распределение ресурсов**
Datora resursu plānošana izpildāmajiem

un rindā gaidošajiem darbiem, piem., operatīvās atmiņas, ievadizvades ierīču un palīgatmiņas plānošana darbiem, kas datorā jāizpilda vienlaicīgi. Sk. arī *dynamic resource allocation*.

1345 **Resource Interchange File Format (RIFF)**
**resursu apmaiņas datnes formāts, formāts *RIFF*, *RIFF***
**формат файла обмена ресурсами**

Multivides datu formāts, ko kopīgi izstrādājušas firmas *IBM* un *Microsoft* un kas nodrošina datņu apmaiņu starp dažādām aparatūras platformām. Sk. arī *media control interface*.

1346 **resource management**
**resursu pārvaldība**
**управление ресурсами**

Darbību kopums, kas aizsargā viena uzdevuma risināšanai izmantojamos resursus no mēģinājuma tos vienlaicīgi izmantot vairāku laiksakritīgu darbu izpildei.

1347 **response time**
**atbildes laiks**
**время ответа**

Laika intervāls, kas sākas tad, kad dators saņem lietotāja pieprasījumu, un beidzas ar pieprasījuma izpildi vai ziņojumu, ka pieprasījums nav izpildāms.

1348 **restart**
**restarts**
**рестарт**

1. Programmas darbības atsākšana, izmantojot kontrolpunkta datus, kas ierakstīti atmiņā.
2. Datora darbības atjaunošana pēc anormālās nobeidzes.

1349 **retrieval**
**izguve**
**выборка**

Operāciju, metožu un procedūru kopums, ko izmanto noteiktas tematikas datu atlasei no datu bāzēm vai datnēm.

1350 **Return key**

Sk. *Enter key*.

1351 **reusability**
**atkārtota lietojamība**
**возможность повторного использования**

Programmatūras moduļu īpašība, kas nodrošina to izmantošanas iespējas vairākās datoru programmās vai programmu sistēmās.

1352 **reverse video**
**reversais video**
**негативное видеоизображение**

Rakstzīmes, laukuma vai kursora izcelšanas paņēmiens vienkrāsas monitorā, mainot vietām to krāsu ar fona krāsu, piem., baltu rakstzīmi uz melna fona aizstājot ar melnu rakstzīmi uz balta fona.

1353 **RFC**

Sk. *Request For Comments*.

1354 **RGB (Red Green Blue) monitor**
**sarkanzaļzilais monitors, *RGB* monitors**
***RGB*-монитор**

Monitors, kas sarkanās, zaļās un zilās krāsas signālus no datora pievada tieši katodstaru lampai, neizmantojot dekodētājus.

1355 **ribbon cable**
**lentkabelis**
**ленточный кабель**
Plakans, plāns, vairākvadu kabelis, kurā vadi izvietoti viens otram blakus vienā plaknē. Lentes kabelis tiek plaši izmantots, lai, piem., izveidotu datora iekšējo funkcionālo bloku savienojumus.

1356 **Rich Text Format (RTF)**
**RTF standarts, standarts *RTF*, *RTF***
**расширенный текстовый формат**
Firmas *Microsoft* izveidots teksta formatēšanas standarts. Pēc šī standarta izveidotā datne ir teksta datne, kurā ietvertas instrukcijas, kas apraksta dokumenta formatēšanu. Šādi kodētu dokumentu var lasīt cita ar standartu *RTF* saderīga tekstapstrādes programma, kā arī to var pārsūtīt pa sakaru kanāliem.

1357 **RIFF**
Sk. *Resource Interchange File Format.*

1358 **right justify**
**lab[ē]jā taisnošana**
**выравнивать вправо**
Sk. *flush right.*

1359 **ring network**
**gredzentīkls**
**кольцевая сеть**
Datora tīkla topoloģijas variants, kurā katrs tīkla mezgls (stacija) savienots ar diviem citiem mezgliem, veidojot noslēgtu gredzenu. Šādā tīklā ziņojumus pārsūta vienā virzienā. Katrs mezgls pārbauda pienākušā ziņojuma adresi. Ja ziņojuma adrese sakrīt ar mezgla adresi, tad mezgls to pieņem un apstrādā. Pretējā gadījumā mezgls reģenerē signālu un nodod ziņojumu tīklā pārsūtīšanai nākamajam mezglam. Signālu reģenerācija nodrošina ziņojumu pārsūtīšanu lielākos attālumos nekā tas iespējams zvaigžņtīklos vai maģistrāles tīklos.

1360 **RIP**
Sk. *raster image processor.*

1361 **RISC**
Sk. *Reduced Instruction Set Computer.*

1362 **RLE**
Sk. *run-length encoding.*

1363 **RLL**
Sk. *run-length limited.*

1364 **rollback**
**atrite**
**откат**
Datu bāzes pārvaldības sistēmas spēja anulēt pēdējo transakciju un atgriezt datu bāzi tās iepriekšējā stāvoklī, ja kāda kļūme pārtrauc transakcijas izpildi.

1365 **ROM**
Sk. *Read-Only Memory.*

1366 **ROM BIOS**
**ievadizvades pamatsistēmas lasāmatmiņa, *BIOS* lasāmatmiņa**
**постоянное запоминающее устройство базовой системы ввода-вывода**
Lasāmatmiņa, kurā ierakstītas programmas un instrukcijas personālā datora perifērijas ierīču ievadizvades operāciju veikšanai.

1367 **ROM cartridge**
**lasāmatmiņas kasetne**
**кассета постоянного запоминающего устройства**
Konstruktīvi noslēgts modulis, ko var ievietot datorā un kas satur lasāmatmiņas mikroshēmā ierakstītus datus, fontus vai īpašas programmas.

1368 **root directory**
**saknes direktorijs**
**корневой справочник**
Hierarhiskas datņu sistēmas sākuma direktorijs — ieejas punkts direktoriju "kokā". No šī saknes direktorija atzarojas dažādi direktoriji un apakšdirektoriji. Katrā no tiem var būt viena vai vairākas datnes vai apakšdirektoriji.

1369 ***router***
**maršrutētājs**
**маршрутизатор**
Datu pārraides tīkla ierīce, kas nodrošina datu pārraides maršruta izvēli tīklos, kuriem var būt atšķirīga arhitektūra un protokoli. Sk. arī *brouter*, *gateway*.

1370 **row**
**rinda [tabulā]**
**строка**
Horizontāla izklājlapas šūnu virkne, kas šķērso visu izklājlapu. Datu bāzēs ar rindu parasti saprot ierakstu.

1371 **RS-232-C**
**saskarne *RS-232-C*,**
***RS-232-C***
**интерфейс *RS-232-C***
ASV elektroniskās rūpniecības uzņēmumu apvienības izstrādāts saskarnes standarts, kas paredzēts datu pārraidei, izmantojot seriālās pieslēgvietas. Vairums personāļo datoru ir apgādāti ar *RS-232-C* saderīgām seriālajām pieslēgvietām, ko var izmantot modemu, printeru, skeneru un citu perifērijas ierīču pievienošanai.

1372 **RTF**
Sk. *Rich Text Format*.

1373 **rule**
**robežsvītra; kārtula**
**правило**
1. Tekstapstrādē — taisna līnija, kas norobežo kādu teksta daļu vai grafisku attēlu no pārējā lappuses satura. Svītras biezumu parasti mēra punktos (1 punkts ir 1/72 collas).
2. Ekspertsistēmās — priekšraksts, ko izmanto, lai pārbaudītu attiecīgā objekta *darbības priekšnosacījumus*, *prognozētu* tā darbību noteiktā situācijā un izdarītu attiecīgus secinājumus.

1374 **ruler**
**mērjosla**
**масштабная линейка**
Tekstapstrādes un datorizdevniecības programmās — displeja ekrāna josla, kas collās vai citās mērvienībās norāda rindiņu platumu, rindkopu atkāpes un citu teksta elementu izvietojumu.

1375 **run**
**palaišana; izpilde**
**выполнение**
1. Programmas izpildes uzsākšana.
2. Vienreizēja, nepārtraukta programmas izpildīšana.

1376 **run around**
**apliece**
**обтекание**

Lappuses kompozīcija, kuru veidojot teksts tiek izvietots apkārt ilustrācijām vai grafiskiem attēliem.

**1377 run-length limited (RLL) izpildblīvējuma tehnoloģija, *RLL* tehnoloģija**
**технология *RLL***

Ierakstes tehnoloģija, kas, salīdzinot ar *FM* un *MFM* tehnoloģiju, nodrošina aptuveni par 50% blīvāku informācijas ierakstīšanu magnētiskajā diskā.

**1378 runtime version izpildlaika versija**
**версия периода прогона**

Operētājsistēmas vai sistēmprogrammas vienkāršota versija, kas saistīta ar noteiktu lietojumprogrammu, kura bez šīm sistēmprogrammām nevar tikt izpildīta.

## S

**1379 safety nebīstamība**
**безопасность**

Sistēmas īpašība nekad nekaitēt tās lietotājam vai apkārtējai videi. Nebīstamības pakāpe atkarīga no sistēmas funkcionēšanas un lietojuma īpatnībām. Tā piem., nebīstamības prasības lidaparāta vadības sistēmai parasti ir ievērojami augstākas nekā stacionāri novietota darbgalda vadības sistēmai.

**1380 sans-serif [character] bezrēdzes [rakstzīme], bezserifa [rakstzīme]**
**сансериф**

Burtveidols, kurā rakstzīmēm nav serifu — īso svītriņu vai ornamentu burtu vilkumu augšējos vai apakšējos galos.

**1381 SAP**

Sk. *Sevice Advertising Protocol.*

**1382 scalable font mērogojams fonts**
**масштабируемый шрифт**

Ekrāna vai printera fonts, kura izmērus var palielināt vai samazināt. Kā mērogojama fonta piemēru var minēt kontūrfontu, ekrāna fontu u.c.

**1383 scalar processor skalārais procesors**
**скалярный процессор**

Procesors, kas paredzēts ātru aprēķinu veikšanai ar skalārajiem mainīgajiem. Sk. arī *vector processor.*

**1384 Scalar Processor Architecture (SPARC) skalārā procesora arhitektūra, arhitektūra *SPARC***
**архитектура *SPARC***

Firmas *SUN Microsystems* 1988. g. izstrādāta atvērta *RISC* datora arhitektūra, kuras centrālā funkcionālā bloka izveide aparatūrā ir relatīvi vienkārša. *RISC* datora izstrādāšanai nav jāizmanto sevišķi augstas pakāpes integrācijas tehnoloģija un samērā viegli var nodrošināt maksimālu tā ātrdarbību. Uz šīs arhitektūras bāzes izveidoti, piem., procesori *Micro SPARC*, *Super SPARC*, *Hyper SPARC*, ko lieto dažādas klases darbstacijās un serveros.

**1385 scaling mērogošana**
**масштабирование**

1. Grafiskā attēla palielināšanas vai samazināšanas process datorgrafikā, piem.,

fonta vai kāda izveidota zīmējuma lieluma maiņa.
2. Decimālā komata vietas noteikšana skaitļiem ar fiksētu vai peldošu komatu.

1386 **scanner**
**skeneris**
**сканер**
Perifērijas ierīce, kas secīgi caurskata un lasa tekstus, attēlus un svītrkodus, pārveido tos ciparu kodā un uzglabā kā datnes, ko ar lappušu izkārtojuma un datorizdevniecības programmu starpniecību var ievietot sagatavojamā dokumenta tekstā. Sk. arī *bit-mapped graphics*, *tagged image file format*.

1387 **schema**
**shēma**
**схема**
Strukturāls datu bāzes apraksts, kurā ir visu lauku tipi, kā arī informācija par katru rindiņu datu bāzes tabulā. Izmantojot valodu *SQL*, shēmas veidošanā tiek iesaistīti visi datu bāzes objekti (tabulas un skatījumi).

1388 **scissoring**
**nošķērēšana**
**отсечение**
Attēla daļas nogriešana, ja tā iziet ārpus noteiktā attēla laukuma.

1389 **Scrapbook**
**datne *Scrapbook***
**файл *Scrapbook***
*Macintosh* saimes personālajos datoros — sistēmas datne, kurā ierakstīti bieži izmantoti teksta un grafikas objekti, piem., firmas vēstuļu augšmalā iespiestais uzraksts.

1390 **screen buffer**
**ekrānbuferis**
**экранный буфер**
Videobuferis ar krāsu videoadapteri aprīkotajos personālajos datoros. Ekrānbuferī parasti ir līdz astoņām lappusēm informācijas, ko var parādīt displeja ekrānā. Sk. arī *active page*, *visual page*.

1391 **screen capture**
**ekrāna notveršana**
**захват экрана**
Ekrāna attēla saglabāšana teksta vai grafikas datnes veidā datora diskatmiņā.

1392 **screen dump**
**ekrāna izmete**
**разгрузка экрана**
Displeja ekrāna attēla (satura) pārsūtīšana printerim drukāšanai.

1393 **screen flicker**
**ekrāna mirgoņa**
**мерцание экрана**
Sk. *flicker*.

1394 **screen font**
**ekrāna fonts**
**экранный шрифт**
Bitkartēts fonts, ko izmanto teksta attēlošanai datora ekrānā.

1395 **screen saver**
**ekrānsaudzētājs**
**предохранитель экрана**
Programmatūra, kas displeja ekrānā ar dažādiem kustīgiem objektiem aizvieto attēlu, lai aizsargātu ekrānu no bojāšanās, ja lietotājs ieslēgto datoru kādu laiku neizmanto.

1396 **script**
**skripts**
**сценарий**
Instrukciju virkne, kas nosaka, kā programmai jāveic kāda specifiska procedūra, piem., ieiešana elektroniskā pasta sistēmā. Dažādās programmās skripta iespējas ir jau iebūvētas, dažus skriptus veido automātiski, pierakstot, kādus taustiņus un komandu izvēlnes lietotājs *izmanto* procedūru laikā. *WWW* kontekstā skripts ir programma, kas atrodas kādā *Web* serverī un apstrādā no pārlūkprogrammas saņemtos pieprasījumus.

1397 **scroll bar**
**ritjosla**
**линейка прокрутки**
Taisnstūrveida josla loga ietvara labajā vai apakšējā malā, ko izmanto loga satura pārvietošanai horizontālā vai vertikālā virzienā.

1398 **scroll box**
**ritlodziņš**
**ползунок линейки прокрутки**
Ritjoslas daļa, kas rāda redzamās informācijas novietojumu attiecībā pret kopējo logā piejamo informāciju. Noklikšķinot peli ritlodziņā un manipulējot ar to, lietotājs var iepazīties ar konkrētajā brīdī logā neredzamo informāciju.

1399 **Scroll Lock key**
**ritslēga taustiņš**
**клавиша блокировки прокрутки**
Firmas *IBM* un ar to saderīgas tastatūras taustiņš, kas parasti izvietots tastatūras augšpusē un darbojas kā slēdzis. Ieslēgtā stāvoklī ritslēga taustiņš nosaka kursora vadīšanas taustiņu darbību. Tā, piem., dažās tekstu rediģēšanas programmās, ja ritslēga taustiņš ir ieslēgts, kursora vadības taustiņi pārvieto kursoru, bet ja izslēgts, — dokumenta saturu, nepārvietojot kursoru.

1400 **scrolling**
**ritināšana**
**прокрутка**
Uz displeja ekrāna attēlotās informācijas pārbīde vertikālā vai horizontālā virzienā. Informācijas attēlojumam pazūdot vienā ekrāna pusē, pretējā pusē parādās jauna informācija vai arī rodas brīva vieta tās ievadīšanai.

1401 **SCSI**
Sk. *Small Computer System Interface.*

1402 **SDK**
Sk. *software developer's kit.*

1403 **search and replace**
**meklēt un aizstāt**
**искать и заменять**
Noteiktu datu vai rakstzīmju secības atrašana un to aizstāšana ar citu datu kopu vai rakstzīmju secību visā dokumentā.

1404 **search engine**
**meklētājdzinējs**
**поисковая машина**
Programma, kas atrod nepieciešamās informācijas izvietojumu datu bāzē. Šo terminu bieži lieto, lai apzīmētu programmu, kas tīklā *Internet* atvieglo lietotājam vajadzīgās informācijas meklēšanu.

1405 **search key**
**meklēšanas atslēga**
**поисковый ключ**
Noteikts lauks ierakstā vai kolonnā, ko

meklē datu bāzē, vai arī vērtība, kas ir atrodama dokumentā vai citā datu kopumā. Sk. arī *alternate key*, *candidate key*, *major key*, *minor key*, *primary key*, *secondary key*.

1406 **secondary cache**
**sekundārā kešatmiņa**
**вторичная кэш-память**

Kešatmiņa, kas atšķirībā no primārās kešatmiņas ir izvietota uz mātesplates, nevis iebūvēta mikroprocesorā. Sekundārā kešatmiņa ievērojami uzlabo sistēmas veiktspēju un ir būtiska jebkuras datoru sistēmas sastāvdaļa.

1407 **secondary key**
**sekundārā atslēga**
**вторичный ключ**

Lauks, kas tiek meklēts tādu ierakstu apakškopā, kuriem ir vienādas primāro atslēgu vērtības. Sk. arī *alternate key*, *candidate key*, *minor key*.

1408 **secondary storage**
**sekundārā atmiņa**
**вспомогательная память**

Datorā energoneatkarīgā atmiņa, kam atšķirībā no primārās (operatīvās) atmiņas ir lielāka ietilpība, bet arī lielāks pieejas laiks. Šajā atmiņā ietilpst tādas atmiņas ierīces, kā, piem., cietie diski, disketes, optiskie diski. Sk. arī *primary storage*.

1409 **secondary window**
**sekundārais logs**
**вспомогательное окно**

Loga veids, kas vienmēr saistīts ar primāro logu un ko var pārvietot vai arī kam var mainīt tā lielumu. Sk. arī *pop-up window*.

1410 **security**
**drošība**
**безопасность**

Datoru sistēmu un tajās glabājamo datu aizsardzība pret to bojāšanu vai zaudēšanu. Galvenā datoru (īpaši daudziem lietotājiem pieejamu datoru sistēmu) drošības problēma ir to aizsardzība pret nesankcionētu izmantošanu.

1411 **segmented address space**
**segmentētā adrešu telpa**
**сегментированное адресное пространство**

Adrešu telpa, kas loģiski sadalīta segmentos. Lai adresētu kādu atmiņas vietu, programmai jāuzdod segments un attālums no segmenta sākumpunkta. Pretstats ir izplātā adrešu telpa.

1412 **selected command**
**atlasītā komanda**
**выбранная команда**

Komanda, ko izvēlas ar peli vai tastatūru un ko izpilda, ja izvēle tiek apstiprināta ar peles dubultklikšķi vai taustiņa nospiešanu.

1413 **selection**
**atlase**
**выбор**

Formatēšanai vai rediģēšanai izgaismota teksta vai grafikas daļa displeja ekrānā.

1414 **selection cursor**
**atlases kursors**
**курсор выбора**

Vizuāla norāde uz pašreizējo pozīciju logā, dialoglodziņā vai izvēlnē.

1415 **selection field**
**atlases lauks**
**поле выбора**

Saistītu izvēļu kopa. Sk. arī *entry field*.

1416 **Sequenced Packet Exchange (SPX) protokols "Secīgā pakešu apmaiņa", protokols *SPX* протокол *SPX***

Viens no firmas *Novell* oprētājsistēmu *NetWare* izmantotajiem protokoliem, kas daudzslāņu arhitektūrā atrodas virs protokola *IPX* un veido transporta slāņa saskarni. Protokols *SPX* nodrošina paketes nogādes pareizības pārbaudi un, ja konstatēta kļūda, tās atkārtotu pārsūtīšanu, kā arī brīdina tīkla operatoru par iespējamām kļūdām savienojumu veidošanā, ja paketes atkārtota pārsūtīšana ir nesekmīga.

1417 **sequential access secīgā pieeja последовательный доступ**

Informācijas uzglabāšanas un izguves metode. Secīgās pieejas izmantošana saistīta ar nepieciešamību pakāpeniski caurskatīt atmiņā esošos datus, lai atrastu meklētos. Atšķirībā no brīvpieejas šādas caurskates nepieciešamība palēnina vajadzīgo datu atrašanu.

1418 **Serial Line Internet Protocol (SLIP) tīkla *Internet* seriālās līnijas protokols, protokols *SLIP*, *SLIP* протокол *SLIP***

Viens no diviem tīkla *Internet* standartprotokoliem, kas, izmantojot telefona līnijas, nodrošina atsevišķas darbstacijas vai datora tiešu pieslēgšanu tīklam. Šis protokols izstrādāts agrāk par protokolu *PPP* un tā iespējas, salīdzinot ar protokolu *PPP*, ir ierobežotas.

1419 **serial mouse seriālā pele последовательная мышь**

Pele, ko pievieno datoram, izmantojot seriālo pieslēgvietu. Atšķirībā no kopnes peles tās izmantošanai nepieciešama brīva seriālā pieslēgvieta.

1420 **serial port seriālā pieslēgvieta последовательный порт**

Ievadizvades pieslēgvieta, kas nodrošina secīgu (bitu pēc bita) informācijas pārsūtīšanu starp datoru un tā ārējām ierīcēm (piem., modemiem, pelēm, skeneriem, seriāliem printeriem u.c.).

1421 **serial printer seriālais printeris последовательный принтер**

Printeris, ko pievieno datora seriālajai pieslēgvietai.

1422 **serif rēdze, serifs сериф**

Īsas svītriņas vai ornamenti, kas pievienoti burtveidola rakstzīmju vilkumu galos.

1423 **server serveris сервер**

Jebkurš dators, kas nodrošina datoru tīkla lietotājiem pieeju datnēm, printeriem, komunikāciju sistēmām vai citiem pakalpojumiem. Parasti serverim ir modernāks procesors, lielāka atmiņa un plašākas kļūdu labošanas iespējas nekā atsevišķa lietotāja darbstacijai. Sk. arī *file server*, *print server*.

1424 **service pakalpojums, serviss сервис**

Lietotājorientētu funkciju kopums, piem., datoru vai programmatūras pircēju teh-

niskā apkalpošana vai datoru tīkla pakalpojumu sniegšana lietotājam. Šaurākā nozīmē šo terminu lieto, lai apzīmētu pakalpojumus, ko kāda programma sniedz citai programmai, lai veicinātu tās izpildi.

1425 **Service Advertising Protocol (SAP)**
**pakalpojumu izziņošanas protokols, protokols *SAP*, *SAP***
**протокол объявления об услугах**

Protokols, kas lokālajos tīklos, kuri strādā oprētājsistēmas *NetWare* vadībā, nodrošina iespēju informēt visus tīkla klientus par serveru sniegtajiem pakalpojumiem

1426 **session**
**seanss; sesija**
**сеанс**

1. Laika intervāls, kurā interaktīvā režīmā izpildāmā programma saņem sākotnējo informāciju, apstrādā to un sniedz atbildi uz lietotāja jautājumiem.
2. Laika intervāls, kurā datu pārraides sistēmās (t.sk. datoru tīklos) notiek datu apmaiņa starp diviem datoriem vai datoru un tā termināli.

1427 **setup**
**uzstādīšana; uzstādījums**
**установка; размещение**

Tāda programmatūras un datora konfigurācijas izveidošana, kas var nodrošināt noteikta veida uzdevumu atrisināšanu.

1428 **setup program**
**uzstādīšanas programma**
**программа установки**

Programmatūra datoru sistēmas izveidošanai noteiktai uzdevumu risināšanas apkārtnei.

1429 **SGML**

Sk. *Standard Generalized Markup Language*.

1430 **shadow memory**
**ēnatmiņa**
**теневая память**

Brīvpieejas atmiņas apagabals, ko izmanto, lai tajā iesūtītu un izpildes laikā saglabātu programmatūru (piem., sistēmu *BIOS*), ko pastāvīgi glabā lēnākas darbības lasāmatmiņā. Ēnatmiņas izmantošana paaugstina datora veiktspēju. Sk. arī *shadow RAM*.

1431 **shadow RAM**
**brīvpieejas ēnatmiņa**
**теневое запоминающее устройство с произвольной выборкой**

Uz mikroprocesoru *Intel 80386* vai *Intel 80486* bāzes izveidotu personālo datoru augšējās atmiņas apgabals starp 64 kilobaitiem un 1 megabaitu. Šī atmiņas daļa tiek rezervēta programmām, ko parasti uzglabā lasāmatmiņā. Tā kā brīvpieejas atmiņa ir ātrdarbīgāka par lasāmatmiņu, tad brīvpieejas ēnatmiņas izmantošana uzlabo datora veiktspēju. Sk. arī *ROM BIOS*.

1432 **shared resource**
**koplietojams resurss**
**общий ресурс**

1. Resurss, ko datu apstrādei kopīgi izmanto vairākas darbstacijas.
2. Direktoriji, datnes, programmas, kā arī modemi, printeri, ploteri un citas ievadizvades ierīces, ko kopīgi izmanto datoru tīkla lietotāji.

1433 **shareware**
**izplatāmprogrammatūra**
**условно-бесплатные программы**

Programmatūra, kuru, lai pārbaudītu tās lietojamību, izplata bez maksas. Ja šo programmatūru grib izmantot regulāri, par to jāmaksā. Lietotājs par šo maksu parasti saņem papildus dokumentāciju, konsultācijas un nākošo bezmaksas versiju.

1434 **shell**
**čaula**
**оболочка**

Programmatūras funkcionāli ārējais slānis (papildprogrammas). Tas nodrošina izvēļņvadāmu vai ikonorientētu lietotāja saskarni ar sistēmu un atvieglo tās lietošanu. Čaulas parasti izmanto komandvadāmās operētājsistēmās, piem., operētājsistēmās *DOS*, *UNIX*, *Microsoft Windows* u.c. Sk. arī *Program Manager*.

1435 **Shift key**
**pārslēgšanas taustiņš**
**клавиша переключения регистра**

Tastatūras taustiņš, kurš nospiestā stāvoklī piešķir pārējiem taustiņiem citu nozīmi, piem., lielā burta ievadi, ja tiek nospiests burta taustiņš, vai simbola ievadi, ja tiek nospiests ciparu taustiņš. Bez tam pārslēgšanas taustiņš bieži tiek izmantots kombinācijās ar citiem taustiņiem, lai ievadītu nestandarta rakstzīmes vai izpildītu citas speciālas funkcijas.

1436 **shortcut**
**īsinājumikona**
**икона оперативного доступа**

Ikona, ko izmanto operētājsistēma *Microsoft Windows 95* un kas nodrošina ātru pieeju kādai programmai. Programmas izpildi uzsāk ar dubultklikšķi uz šīs programmas ikonas.

1437 **shortcut key**
**īsinājumtaustiņš**
**сокращенная комбинация клавиш**

Taustiņu kombinācija, kas ar vienu taustiņa piesitienu nodrošina kādas komandas izpildi vai pieeju dialoglodziņam, apejot izvēļņu hierarhiju.

1438 **sidelit**
**sāngaismojums**
**боковая подсветка**

Šķidro kristālu apgaismošanas veids, ko parasti izmanto piezīmjdatoros un klēpjdatoros, iebūvējot papildus gaismas avotu ekrāna sānos. Sk. arī *backlit*.

1439 **signature**
**signatūra**
**сигнатура**

Speciāls autentifikācijas kods, ko lietotājs ievada pirms sistēmas izmantošanas vai uzdevuma izpildes, lai pierādītu savu identitāti (piem., parole).

1440 **signature file**
**signatūru datne**
**файл сигнатур**

Datne, kuru automātiski pievieno nosūtāmajam elektroniskā pasta ziņojumam un kurā ir ziņojuma nosūtītāja adrese, elektroniskā pasta adrese un var būt arī telefona un faksa numurs. Ieteicamais šīs datnes apjoms — 3—5 rindiņas.

1441 **SIMM**

Sk. *single in-line memory module*.

1442 **Simple Mail Transfer Protocol (SMTP)**
**vienkāršais pasta pārsūtīšanas protokols, protokols *SMTP*, *SMTP***
**протокол *SMTP***

Protokolu *TCP/IP* sistēmas elements elektroniskā pasta parsūtīšanai. Izmantojot protokola *TCP* pakalpojumus, protokols *SMTP* nosaka starp tīkla serveriem pārsūtāmo tekstu struktūru un attiecīgās elektroniskā pasta vadības procedūru.

1443 **Simple Network Management Protocol (SNMP)**
**vienkāršais tīkla pārvaldības protokols, protokols *SNMP*, *SNMP***
**протокол *SNMP***

Protokolu *TCP/IP* sistēmas sastāvdaļa, kas, izmantojot protokola *UDP* pakalpojumus, veic tīkla mezglu un iekārtu (piem., maršrutētāju) pārvaldību un konfigurācijas pārraudzību. Protokols *SNMP* nosaka priekšrakstus attiecīgās dienesta informācijas pārraidei starp vadošo un vadāmo objektu. Sk. arī *client/server architecture*.

1444 **simulation**
**imitācija, simulācija**
**имитационное моделирование**

Parādību un procesu pētīšanas metode, kurā pētāmo objektu aizstāj ar kādu citu sistēmu (modeli). Piemēram, pētāmo objektu aizstāj ar tā matemātisko aprakstu un imitāciju veic analītiski. Bieži izmanto dažādus imitēšanas veidus, kas balstās uz sistēmas darbības aprakstu ar datora programmu.

1445 **single board computer (SBC)**
**vienplates dators, vienplates skaitļotājs**
**одноплатный компьютер**

Mikrodators vai minidators, kura visas komponentes ir izvietotas uz vienas drukātās shēmas plates.

1446 **single density disk**
**vienblīvuma diskete**
***дискета с одинарной плотностью***

Diskete, kurā informācijas ierakstīšanu izdara, izmantojot *FM* tehnoloģiju, un kurā ierakstītās informācijas blīvums ir salīdzinoši mazs. Sk. arī *double-density disk, high-density disk*.

1447 **single in-line memory module (SIMM)**
**vienrindas atmiņas modulis, modulis *SIMM***
**модуль с однорядным расположением микросхем памяти**

Šaura drukātās shēmas plate, uz kuras vienā rindā parasti izvietotas 8 vai 9 lasāmatmiņas mikroshēmas. Moduli *SIMM* ar speciālu šķautņspraudņu starpniecību pievieno datora mātesplatei.

1448 **single in-line package (SIP)**
**vienrindas korpuss, *SIP* korpuss**
**однорядовый корпус**

Plakans elektronisko komponentu konstruktīvs elements, kas izveidots līdzīgi modulim *SIMM*, bet tajā drukāto šķautņspraudņu vietā izvadi izveidoti kā kontaktkājiņas vienā no garākajām korpusa malām. Sk. arī *dual in-line package*.

1449 **single-sided disk**
**vienpusēja diskete; vienpusējs disks**
**одностороняя дискета**
Diskete vai cietais disks, kurā datus var ierakstīt un uzglabāt tikai vienā tās (tā) pusē.

1450 **SIP**
Sk. *single in-line package.*

1451 **site licence**
**vietu licence**
**лицензия на использование системы**
Programmatūras izplatītāja un pircēja juridiska vienošanās, kas nosaka kopiju skaitu, ko pircējs var izmantot iekšējai lietošanai.

1452 **sleep mode**
**miega režīms**
**режим ожидания**
Ar barošanas pārvaldības līdzekļiem aprīkotu datoru darbības režīms, kurā atsevišķi komponenti, kas netiek izmantoti, uz laiku tiek atslēgti. Lai ietaupītu elektroenerģiju, kas nepieciešama atmiņas mikroshēmas atsvaidzināšanai, šajā režīmā operatīvās atmiņas saturs parasti tiek ierakstīts cietajā diskā. Sk. arī *display power management system, green PC.*

1453 **slide show**
**slīdrāde**
**слайдовая презентация**
Iepriekš izvēlētu diagrammu un grafiku izspīdināšana displeja ekrānā noteiktā kārtībā. Pievienojot papildus spiežampogas, var mainīt programmā paredzēto slīdu parādīšanas kārtību, parādīt kādu konkrētu slīdu vai pārtraukt slīdu demonstrēšanu.

1454 **SLIP/PPP**
Sk. *Serial Line Internet Protocol*; *Point-to-Point Protocol.*

1455 **slot**
**slots**
**слот**
Sk. *expansion slot.*

1456 **Small Computer System Interface (SCSI)**
**mazo datorsistēmu saskarne, saskarne *SCSI***
**интерфейс малых вычислительных систем, интерфейс *SCSI***
Amerikas Nacionālā standartu institūta akceptēta ātrdarbīgas paralēlās saskarnes standartspecifikācija, kas nosaka, kādā veidā cietie diski, printeri, optiskie diski un citas ierīces pieslēdzamas personālajiem datoriem.

1457 **smart cable**
**viedais kabelis**
**интеллектуальный кабель**
Divu ierīču savienotājkabelis, kurā ir iebūvēts mikroprocesors, kas analizē ienākošos signālus un pārveido tos atbilstoši attiecīgā protokola prasībām.

1458 **smart card**
**viedkarte**
**интеллектуальная карта**
1. Drukātās shēmas plate ar iebūvētu loģiku vai mikroprocesoru, kas tai nodrošina zināmas lēmumu pieņemšanas iespējas.
2. Plastmasas karte ar iebūvētu atmiņu un mikroprocesoru, ko izmanto lietotāju identificēšanai vai naudas pārvedumiem.

1459 **smart terminal**
**viedais terminālis**
**интеллектуальный терминал**
Vairāklietotāju sistēmu terminālis, kuram ir savas datu apstrādes ierīces un ar kuru var ne tikai izgūt datus no resursdatora, bet arī patstāvīgi veikt datu apstrādi. Sk. arī *dumb terminal*.

1460 **smoothing**
**gludināšana**
**сглаживание**
Dažu lāzerprinteru izmantotā tehnika līkņu nogludināšanai, ko īsteno, samazinot līniju veidojošo punktu lielumu.

1461 **SMTP**
Sk. *Simple Mail Transfer Protocol*.

1462 **snapshot**
**momentuzņēmums**
**моментальный снимок**
Galvenās atmiņas vai videoatmiņas kopija, ko fiksē noteiktā laika brīdī un pārsūta uz printeri vai cieto disku. Momentuzņēmumi tiek periodiski atkārtoti, lai atteices gadījumā atjaunotu sistēmu.

1463 **SNMP**
Sk. *Simple Network Management Protocol*.

1464 **snow**
**ņirba**
**снежок**
Mazu mirgojošu punktiņu parādīšanās uz ekrāna, kad attēli displeja ekrānā mainās ātrāk nekā displeja aparatūra tos var apstrādāt.

1465 **soft copy**
**mīkstā kopija**
**недокументальная копия**
Elektroniska informācijas uzglabāšanas vai izvades forma kādā no personālajā datorā izmantojamajām datu vidēm, piem., disketē, cietajā diskā, lasāmatmiņas kompaktdiskā, magnētiskajā lentē, displeja ekrānā. Sk. arī *hard copy*.

1466 **soft font**
**nestingrais fonts**
**мягкий шрифт**
Sk. *downloadable font*.

1467 **soft return**
**nestingrā atgrieze**
**мягкий возврат**
Teksta rindiņas pārtraukšana, ko veic teksta apstrādātājs, ja nākošais vārds teksta rindiņā iesniedzas lappuses malā. Kad dokuments tiek drukāts, nestingrā atgrieze tiek pārvērsta printera rindiņas beigu kodā.

1468 **software**
**programmatūra**
**программное обеспечение**
Datoru programmas, procedūras un ar tām saistītā dokumentācija un dati, kas nepieciešami datoru sistēmas darbībai. Sk. arī *hardware*, *application software*, *software product*, *support software*, *system software*.

1469 **software cache**
**programmatūras kešatmiņa**
**программная кэшпамять**
Liels brīvpieejas atmiņas apgabals, kurā ar speciālu programmu starpniecību tiek glabāti bieži lietojami dati un programmas. Pietiekami liela apjoma (1-2 megabaitu) programmatūras kešatmiņa var būtiski paātrināt diskatmiņas darbību, izmantojot, piem., datu bāzu pārvaldības programmas. Sk. arī *hardware cache*, *primary cache*, *secondary cache*.

1470 **software developer's kit (SDK)**
**programmatūras izstrādātāja aprīkojums**
**набор инструментальных средств разработчика программного обеспечения**

Sk. *developer's kit.*

1471 **software engineering**
**programminženierija**
**программотехника**

Sistematizēta pieeja zinātnisko un tehnoloģisko zināšanu, metožu un pieredzes izmantošanai funkcionāli efektīvas programmatūras izstrādāšanas, testēšanas un ieviešanas procesā. Sk. arī *programming.*

1472 **software handshaking**
**programmsarokošanās**
**программное подтверждение связи**

Datu plūsmas vadības metode, kas nodrošina, ka viena modema nosūtītie dati nepārplūdina otru modemu, ar kuru notiek datu apmaiņa. Dažās programmatūras sarokošanās shēmās paredzēta speciālu kodu apmaiņa starp modemiem tajā brīdī, kad tie ir gatavi nosūtīt vai saņemt datus.

1473 **software interrupt**
**programmpārtraukums**
**программное прерывание**

Pārtraukums, ko izsauc kāda instrukcija programmā, kura aptur notiekošo informācijas apstrādes procesu, lai pieprasītu pārtraukuma apdarinātāja pakalpojumus, piem., organizētu datu ievadi vai izvadi. Sk. arī *hardware interrupt.*

1474 **software licence**
**programmatūras licence**
**лицензия на программное обеспечение**

Lietotāja un programmatūras izstrādātāja juridiska vienošanās, kas nosaka programmatūras lietotāja tiesības un pienākumus, kā arī programmatūras izplatītāja atbildību. Sk. arī *site licence.*

1475 **software life cycle**
**programmatūras dzīves cikls**
**жизненный цикл программного обеспечения**

Viss programmatūras pastāvēšanas laiks — no tās izstrādāšanas sākuma līdz brīdim, kad tā ir zaudējusi savu praktisko vērtību. Programmatūras dzīves cikla galvenās fāzes ir projektēšana, izstrādāšana, ekspluatācija un, iespējams, arī modernizēšana.

1476 **software package**
**programmatūras pakotne**
**пакет программ**

Savstarpēji saistītu programmu kopums, kas izveidots kādas noteiktas uzdevumu klases risināšanai un kas ietver visas nepieciešamās palīgprogrammas un programmu dokumentāciju.

1477 **software piracy**
**programmpirātisms**
**нелегальное использование программ**

Nelikumīga darbība, kuras rezultātā bez attiecīgās licences tiek kopēta un izplatīta programmatūra. Sk. arī *copy protection.*

1478 **software product**
**programmatūrasprodukts**
**программный продукт**

Lietotāja vajadzībām izstrādāts pilns datora programmu, procedūru un ar tām saistītās dokumentācijas un datu kopums. Sk. arī *software.*

1479 **software tools**
**programmrīki**
**программный инструментарий**
Programmas, ko izmanto citu programmu izstrādāšanai. Programmrīku sastāvā parasti ir teksta redaktors, kompilators, kā arī ielādes, atkļūdošanas un citas programmas un tos izmanto visā programmatūras izstrādāšanas procesā, ieskaitot izstrādāšanas vadību un kvalitātes nodrošināšanu.

1480 **solid color**
**tīrtoņa krāsa**
**чистый цвет**
Krāsa, ko attēlo displeja ekrānā vai izdrukā bez pustoņiem.

1481 **solid font printer**
**cietfontu printeris**
**принтер сплошного шрифта**
Printeris, kas drukā rakstzīmi vienā paņēmienā, izmantojot tās pilnu formu (piem., ziedlapprinteris).

1482 **sorting**
**kārtošana**
**сортировка**
Objektu kopas pārkārtošana, grupēšana vai atlase pēc noteiktām pazīmēm. Kārtošanu nodrošina operētājsistēmas programmas, kā arī lietojumprogrammas, piem., tekstu apstrādes un datu bāzu vadības programmas.

1483 **sound board**
**skaņas plate**
**звуковая плата**
Sk. *sound card*.

1484 **sound card**
**skaņas karte**
**звуковая карта**
Personālo datoru papildierīce, kas no ciparu datiem ģenerē skaņu atveidojošus analogsignālus, izmantojot vai nu ciparanalogu pārveidotājus, vai frekvences modulēšanas mikroshēmas. Izmantojot ciparanalogu pārveidotāju, skaņas karte parasti nodrošina arī skaņas ieraksti ciparu formā, kā arī vada mūzikas instrumentus (piem., sintezatoru), kas pievienoti datoram, izmantojot saskarni *MIDI*.

1485 **sound file**
**skaņu datne**
**звуковой файл**
Datne, kas glabājas diskatmiņā un kas satur skaņu datus - diskretizētus skaņu analogsignālus vai notis mūzikas instrumentiem, ko pievieno datoram, izmantojot saskarni *MIDI*.

1486 **source**
**avots**
**источник**
Ieraksts, datne, dokuments vai disks, no kura informācija tiek izgūta vai pārsūtīta. Sk. arī *destination*, *source address*.

1487 **source address**
**avota adrese**
**адрес источника**
Ierīces vai atmiņas apgabala adrese, no kuras tiek pārraidīti dati. Sk. arī *destination*, *destination address*, *source*.

1488 **source document**
**pirmdokuments**
**исходный документ**
Dokuments, no kura dati tiek kopēti un izmantoti citos dokumentos.

1489 **source program**
**pirmprogramma**
**исходная программа**

Programmas pieraksts augsta līmeņa programmēšanas valodā, kuru attiecīgais valodas kompilators vai translators pārveido datorā izpildāmā formā.

1490 **space**
**atstarpe; starpa; telpa**
**пробел; пространство**
1. Viena vai vairākas tukšumzīmes.
2. Datu glabāšanai paredzēta vieta datora atmiņā.
3. Tukšs laukums, kas atdala teksta vārdus vai rindiņas.

1491 **space character**
**atstarpes rakstzīme**
**знак пробела**
Rakstzīme, kuru ievada datorā, izmantojot atstarpēšanas taustiņu. Šīs rakstzīmes ievadīšanas rezultātā veidojas atstarpe, drukas vai displeja aktīvajai pozīcijai pavirzoties par vienu vietu uz priekšu.

1492 **Spacebar**
**atstarpēšanas taustiņš**
**клавиша пробела**
Garš taustiņš tatstatūras apakšdaļā, kas ievada atstarpes rakstzīmi.

1493 **SPARC**
Sk. *Scalar Processor Architecture*.

1494 **SPECmark**
Sk. *System Performance Evaluation Cooperative Mark*.

1495 **special character**
**speciāla rakstzīme**
**специальный знак**
Rakstzīme, kura nav ne burts, ne cipars, ne atstarpes rakstzīme. Šāda rakstzīme ir, piem., #, $, %, &, +.

1496 **speech recognition**
**runas pazīšana**
**распознавание речи**
Cilvēka runas datorizēta atpazīšana un tās pārveidošana ciparu formā, kas piemērota uzglabāšanai un apstrādei datorā.

1497 **speech synthesis**
**runas sintēze**
**синтез речи**
Cilvēka runai līdzīga audioizvada veidošana, pārveidojot datorā esošos datus tiem atbilstošās fonēmu kombinācijās.

1498 **speech synthesizer**
**runas sintezators**
**синтезатор речи**
Ierīce, kas, izmantojot ciparinformāciju, ģenerē cilvēka balsij līdzīgas skaņas.

1499 **spelling checker**
**pareizrakstības pārbaudītājs**
**блок орфографического контроля**
Programma, ar kuru pārbauda vārdu pareizrakstību tekstā, salīdzinot katru vārdu ar pareizi uzrakstīto vārdu vārdnīcā, kas glabājas datora diskatmiņā. Pareizi uzrakstītie, bet nepareizā nozīmē lietotie vārdi netiek atklāti.

1500 **split bar**
**skaldītājjosla**
**линия разделения**
Taisnstūrveida josla, kuru pārvietojot logu var sadalīt horizontālās vai vertikālās daļās.

1501 **split screen**
**skaldītais ekrāns**
**разделенный экран**
Attēlošanas veids, kas sadala displeja ekrānu vairākos logos, dažādos logos

attēlojot dažādas datnes vai vienas un tās pašas datnes dažādas daļas. Skaldītā ekrāna izmantošana ir ļoti ērta, ja, gatavojot vienu dokumentu, nepieciešams atsaukties uz otru dokumentu vai arī ja teksta rediģēšanā izmanto izgriešanu un ielīmēšanu.

1502 **spreadsheet**
**izklājlapa**
**электронная таблица**

Darba formulārs, kas displeja ekrānā attēlots kā divdimensiju matrica, kas sastāv no rindām un kolonnām. Rindu un kolonnu krustpunktus sauc par šūnām. Šūnās var ievadīt iezīmes, skaitliskas vērtības un formulas. Sk. arī *spreadsheet program*.

1503 **spreadsheet program**
**izklājprogramma**
**программа табличных вычислений**

Programma, kas nodrošina lietotāja interaktīvu sadarbību ar displeja ekrānu un dod iespēju izmantot uz ekrāna izklājlapā attēlotos skaitliskos datus, formulas un tekstu. Izklājprogrammas galvenokārt izmanto dažādu finansiālu aprēķinu veikšanai, kā arī plānošanai un prognozēšanai. Daudzām izklājprogrammām piemīt plašas grafiskās informācijas veidošanas iespējas.

1504 **sprite**
**gariņš**
**спрайт**

Maza izmēra attēls datorgrafikā, ko var bīdīt pa ekrānu neatkarīgi no citiem fona attēliem. Gariņus parasti izmanto datorspēlēs.

1505 **SPS**

Sk. *standby power system*.

1506 **SPX**

Sk. *Sequenced Packet Exchange*.

1507 **SQL**

Sk. *Structured Query Language*.

1508 **SRAM**

Sk. *Static Random-Access Memory*.

1509 **ST 506**
**saskarne *ST 506***
**интерфейс *ST 506***

Saskarne, kas nosaka, kādā veidā cieto disku diskdzinis pieslēdzams pirmajiem firmas *IBM* un ar tiem saderīgajiem personālajiem datoriem. Saskarne *ST 506* nenodrošina pietiekami lielu datu pārsūtīšanas ātrumu, un to aizvieto ar saskarnēm *EIDE*, *ESDI* un *SCSI*.

1510 **Standard Generalized Markup Language (SGML)**
**vispārinātā marķēšanas standartvaloda, valoda *SGML***
**стандартный обобщенный язык разметки, язык *SGML***

Aparatūrneatkarīga standartizēta formāla valoda, kas nosaka, kā jāveic dokumentu iezīmēšana, lai noteiktu tādu teksta noformēšanas elementu kā treknraksta, slīpraksta, malu u.c. izmantošanu. Šo valodu parasti izmanto, lai kodētu datus datu bāzēs vai izdarītu iezīmes grāmatu tekstā pirms to iespiešanas.

1511 **standby power system (SPS)**
**nodroses barošanas sistēma**
**резервная система питания**

Nepārtrauktās barošanas nodrošinājuma sistēma, kas pieslēdz dublējošās baterijas, ja tiek konstatēts barošanas avota bojājums.

1512 **star network**
**zvaigžņtīkls**
**звездообразная сеть**
Datoru tīkla topoloģijas variants, kas pēc savas formas atgādina zvaigzni. Zvaigznes centrā ir centrmezgls (koncentrators), ko individuāli kabeļi savieno ar pārējiem tīkla mezgliem (stacijām). Sk. arī *bus network*, *ring network*.

1513 **star-dot-star**
**zvaigzne-punkts-zvaigzne**
**звездочка-точка-звездочка**
Operētājsistēmas MS-DOS aizstājējzīme *.*, kas ļauj datoram meklēt datnes, neanalizējot to nosaukumus vai datņu nosaukumu paplašinājumus.

1514 **startup disk**
**starta disks**
**диск запуска**
Disks, kurā ir sistēmas apslēptās datnes un datne *COMMAND.COM*, kas nepieciešama datora darbības uzsākšanai.

1515 **statement**
**priekšraksts**
**утверждение**
Teikums programmēšanas valodā, kas apraksta izpildāmās operācijas un ir sintaktiski un semantiski pilnīgi noformēts. Pēc kompilācijas vai interpretācijas priekšraksts tiek pārveidots vienā vai vairākās datora mašīnvalodas instrukcijās.

1516 **Static Random-Access Memory (SRAM)**
**statiskā brīvpieejas atmiņa, *SRAM* atmiņa**
**статическое запоминающее устройство с произвольной выборкой**
Datora atmiņa, kurā katra informācijas elementa (bita) uzglabāšanai izmanto pusvadītāju trigeri. Atšķirībā no dinamiskās brīvpieejas atmiņas tās saturu nav nepieciešams regulāri atjaunot un tās ātrdarbība ir būtiski lielāka, bet šī tipa atmiņa ir arī ievērojami dārgāka.

1517 **STN (Super Twist Nematic)**
Sk. *supertwist*.

1518 **storage allocation**
**atmiņas iedalīšana**
**распределение памяти**
Darbība, kas atmiņas apgabalu piešķir atsevišķiem procesiem. Vairākprogrammu sistēmā atmiņas iedalīšana ir nepieciešama, lai nodrošinātu procesus pret savstarpējiem traucējumiem. Kopējus atmiņas apgabalus izmanto tad, ja procesi ir savstarpēji saistīti.

1519 **storage device**
**atmiņas ierīce**
**запоминающее устройство**
Jebkura optiska vai magnētiska ierīce informācijas glabāšanai datorā. Sk. arī *primary storage*, *secondary storage*.

1520 **stream-oriented file**
**plūsmorientēta datne**
**потоковый файл**
Datne, kuras struktūra ir atvērtāka nekā parastajai datu datnei (piem., dokumenta teksts, diskrēts balss ieraksts). Teksts un balss ieraksts veido nepārtrauktu rakstzīmju plūsmu atšķirībā no datu bāzes ierakstiem, kam ir fiksēta formāta struktūra un kas atkārtojas.

1521 **streamer**
**strīmeris**
**стример**
Sk. *streaming tape drive*.

1522 **streaming tape drive**
**lentes plūsmdzinis, strīmeris**
**стример**
Lenšdzinis, ko bieži izmanto, lai veidotu cieto disku informācijas dublētājkopijas.

1523 **string**
**virkne**
**цепочка**
Līdzīgas dabas elementu (piem., rakstzīmju vai ierakstu) secība, kas tiek uzskatīta par vienotu veselumu. Sk. arī *character string*.

1524 **stroke**
**vilkums**
**штрих**
Spalvas vai otas vilciena platums (pikseļos), zīmējot attēlu displeja ekrānā, vai drukāto rakstzīmju biezums, ko veido, piem., lāzerprinteris.

1525 **structural testing**
**strukturālā testēšana**
**структурное тестирование**
Testēšana, kura atšķirībā no funkcionālās testēšanas testē arī sistēmas vai tās komponentu iekšējo struktūru.

1526 **structured design**
**strukturētā projektēšana**
**структурное проектирование**
Sistemātiska pieeja programmatūras projektēšanai, ievērojot modularitātes, lejupejošās projektēšanas, sistēmas struktūras un tās funkcionēšanas dalījumu soļos.

1527 **Structured Query Language (SQL)**
**strukturēta vaicājumvaloda, valoda *SQL***
**язык структурированных запросов, язык *SQL***
Firmas *IBM* izstrādāta valoda, ko lieto datu bāzes pārvaldības sistēmās dažāda tipa datoros. Valoda *SQL* tiek izmantota klientservera arhitektūras tīklos, lai nodrošinātu personālajiem datoriem pieeju kopīgi izmantojamu datu bāzu resursiem. Izmantojot šo valodu, lietotājam nav jārūpējas par to, kā fizikāli tiek īstenota pieeja datiem un kā tiek nodrošināta pieeja datu bāzēm, kas izvietotas gan lieldatoros, gan arī minidatoros un personālajos datoros.

1528 **style**
**drukas stils**
**начертание, стиль**
Tekstu sagatavošanā izmantojamo paragrāfu un lappušu izkārtojuma, kā arī malu izmēru, kolonnu platuma, rakstzīmju atribūtu (kursīva, treknraksta, augšraksta, apakšraksta u.c.), fontu un fontu lieluma kopums, kas nosaka teksta stilistisko izveidi.

1529 **style sheet**
**stila lapa**
**таблица стилей**
Tekstapstrādē un datorizdevniecībā — datne ar piemērotiem izkārtojuma veidiem noteiktām dokumentu kategorijām, piem., personiskām vēstulēm, oficiālām vēstulēm, pārskatiem, rokas grāmatām u.c. Stila lapā var uzdot dažādus izklājuma parametrus, piem., lappuses izmērus, malu lielumu, kolonnu platumu, rindkopu atkāpes, fontus un to lielumu.

1530 **stylus**
**irbulis**
**щип**
Rādītājierīce, ko lieto, lai, izmantojot ciparošanas planšeti, zīmētu attēlus displeja ekrānā vai norādītu uz kādu tajā

redzamās izvēlnes elementu. Sk. arī *light pen*.

1531 **subdirectory**
**apakšdirektorijs**
**подсправочник**
Direktorijs, kas iekļauts augstākas hierarhijas direktorijā. Lai organizētu pieeju datnēm apakšdirektorijā, jāuzrāda visi direktoriji, kas hierarhiskajā sistēmā atrodas virs tā.

1532 **submenu**
**apakšizvēlne**
**субменю**
Izvēlne, kas ir kādas citas izvēlnes sastāvdaļa. Pēdējo parasti sauc par galveno vai pamatizvēlni.

1533 **subschema**
**apakšshēma**
**подсхема**
Atsevišķa lietotāja subjektīvs datu bāzes skatījums. Sk. arī *schema*.

1534 **subscript**
**apakšraksts**
**нижний индекс**
Viena vai vairākas rakstzīmes, kas izdrukātas nedaudz zemāk par apkārtējo tekstu un ko parasti izmanto matemātikā un ķīmijā (piem., $n_{ij}$, $l_{min}$, $H_2O$). Sk. arī *superscript*.

1535 **super VGA (SVGA)**
**videostandarts *SVGA*;**
**videoadapteris *SVGA*,**
**adapteris *SVGA***
**видеостандарт *SVGA*;**
**видеоадаптер *SVGA*;**
**адаптер *SVGA***
1. Videostandarts, ko asociācija *VESA* izstrādājusi *IBM PC* tipa datoriem un kas izceļas ar augstu izšķirtspēju un ir uzskatāms par videostandarta *VGA* uzlabojumu.
2. Videoadapteris, kas atbilst videostandartam *SVGA* un, izmantojot papildatmiņu un piemērotu monitoru, nodrošina videostandarta *SVGA*, *VGA*, kā arī *EGA* un *CGA* prasību izpildi. Sk. arī *backward compatible*.

1536 **superscalar processor**
**superskalārs procesors**
**суперскалярный процессор**
Mikroprocesors, kas var vienā taktī izpildīt vairākas instrukcijas. To panāk ar iebūvēta plānotāja palīdzību, identificējot nekonfliktējošu instrukciju grupas, ko var izpildīt vienlaicīgi, izmantojot vairākus centrālos procesorus.

1537 **superscript**
**augšraksts**
**верхний индекс**
Viena vai vairākas rakstzīmes, kas nodrukātas nedaudz augstāk par apkārtējo tekstu (piem., $n^2$, $a^*$). Sk. arī *subscript*.

1538 **supertwist**
**supervijums**
**супертвист**
Kristālu izvietojuma tehnoloģija, ko izmanto šķidro kristālu displejos, lai nodrošinātu plašāku redzes leņķi un uzlabotu attēla kontrastainību.

1539 **support software**
**atbalstprogrammatūra**
**вспомогательное программное обеспечение**
Programmatūra, kas palīdz izstrādāt vai uzturēt citas programmas, piem., kompilatorus, ielādes programmas un citas palīgprogrammas. Sk. arī *software*, *software developer's kit*.

1540 **surfing**
**sērfošana**
**серфинг**
Nelineārs informācijas meklēšanas process tīkla *Internet* kibertelpā.

1541 **SVGA**
Sk. *super VGA*.

1542 **swapping**
**pārnešana**
***перекачка***
1. Operatīvās atmiņas satura pārsūtīšana palīgatmiņā un otrādi.
2. Virtuālās atmiņas gadījumā — jaunas lappuses pārsūtīšana no palīgatmiņas operatīvajā atmiņā iepriekš iesūtītās lappuses vietā.

1543 **syntax**
**sintakse**
**синтаксис**
Noteikumi, kas nosaka programmēšanas valodā atļauto konstrukciju izveidošanu, kā arī rakstzīmju izvietojumu programmā. Sintakse nosaka tikai konstrukciju formu, bet nenosaka to saturu.

1544 **syntax error**
**sintakses kļūda**
**синтаксическая ошибка**
Kļūda, ko izraisījusi izmantotās programmēšanas valodas sintakses noteikumu neievērošana.

1545 **Sys Req key**
Sk. *System Request key*.

1546 **system**
**sistēma**
**система**
Objektu, procedūru vai paņēmienu kopums un to savstarpējās attiecības, kas funkcionāli veido vienotu veselumu.

1547 **system colours**
**sistēmas krāsas**
**системные цвета**
Krāsu palete (parasti 20 krāsas), ko izmanto operētājsistēma *Microsoft Windows*, lai iekrāsotu tādus loga elementus kā malas, titrus, paskaidrojošus uzrakstus, ekrāna pogas u.tml.

1548 **system file**
**sistēmas datne**
**системный файл**
Datne, kas ir operētājsistēmas vai kādas citas vadības sistēmas sastāvdaļa un kas satur informāciju par šai sistēmai nepieciešamajiem resursiem.

1549 **system font**
**sistēmas fonts**
**системный шрифт**
Operētājsistēmā vai grafiskajā lietotāja saskarnē iebūvēts fonts. Šos fontus izmanto kā ekrāna fontus, piem., izvēlņu vai dialoglodziņu virsrakstiem.

1550 ***system palette***
**sistēmas palete**
**системная палитра**
Krāsu palete, kuras izmantošanu dažādiem lietojumiem nodrošina konkrēta operētājsistēma. Sistēmas palete ir atkarīga no izmantotās aparatūras un displeja adaptera tipa.

1551 **System Performance Evaluation Cooperative Mark (SPECmark)**
**sistēmas veiktspējas novērtējuma etalonprogrammu komplekts, etalonprogrammas *SPECmark***
**эталонные программы *SPECmark***

Desmit etalonprogrammu (etalonuzdevumu) komplekts, kas izveidots, lai pārbaudītu datora vai datoru sistēmas veiktspēju risināt uzdevumus ar veselajiem skaitļiem (*SPECint*) un skaitļiem ar peldošo komatu (*SPECfp*). Sk. arī *Dhrystones, Whetstones.*

**1552 system prompt**
**sistēmaș uzvedne**
**системная подсказка**

Komandvadāmās operētājsistēmās — teksts, kas norāda, ka operētājsistēma var veikt tādus uzdevumus kā datņu kopēšanu, disku formatēšanu un programmu ielādi. Sk. arī *command-line operating system.*

**1553 System Request key**
**sistēmpieprasīšanas taustiņš**
**клавиша запроса системы**

Firmas *IBM* un ar to saderīgas tastatūras taustiņš, kas, personālajam datoram strādājot lieldatora termināļa režīmā, ļauj lietotājam pārslēgties no viena seansa uz otru vai veikt speciālas termināļa pārslēgšanas darbības.

**1554 system software**
**sistēmas programmatūra**
**системное программное обеспечение**

Lietojumneatkarīga programmatūra, kas nodrošina lietojumprogrammu izpildi. Pie sistēmas programmatūras pieder, piem., operētājsistēmas, tīklu un datu bāzu pārvaldības programmas.

**1555 system unit**
**sistēmas bloks**
**системный блок**

Personālā datora funkcionāls bloks, kurā parasti ir procesors, atmiņas ierīces, ievadizvades kanāli, kā arī viens vai divi diskešu diskdziņi.

**1556 Systems Application Architecture (SAA) sistēmu lietojumu arhitektūra, arhitektūra *SAA***
**архитектура среды для разработки приложений**

Standartu kopa, kas nosaka sakarus starp dažāda tipa firmas *IBM* datoriem, sākot no personālajiem datoriem līdz lieldatoriem. Sistēmu lietojumu arhitektūru veido trīs sastāvdaļas: vienotais sakaru nodrošinājums, vienotā programmēšanas saskarne, vienotā lietotāja pieejas arhitektūra.


**1557 tab character**
**tabulēšanas rakstzīme**
**символ табуляции**

Vadības rakstzīme, ko vairums teksta redaktoru uztver kā komandu pārvietot kursoru vai drukas galviņu par noteiktu pozīciju skaitu, parasti līdz nākošajai tabulēšanas pieturai. Tabulēšanas rakstzīmes *ASCII* kods ir 09. Tabulēšanas rakstzīmi ievada ar tabulēšanas taustiņa palīdzību.

**1558 Tab key**

Sk. *Tabulator key.*

**1559 tab stop**
**tabulēšanas pietura**
**позиция табуляции**

Teksta rindiņas pozīcija, kurā nonāk kursors vai drukas galviņa pēc tabulēšanas rakstzīmes saņemšanas. Attālums

starp tabulēšanas pieturām operētājsistēmai *DOS* ir 8 pozīcijas, grafiskajos teksta redaktoros tas parasti ir 1/2 collas, bet lietotājs to var arī mainīt.

1560 **tabbing**
**tabulēšana**
**табуляция**

Kursora pārvietošana pa displeja ekrānu vai printera drukāšanas galviņas pārvietošana līdz noteiktai kolonnai apstrādājamajā tabulā. Sk. arī *tabulator key*.

1561 **table**
**tabula**
**таблица**

1. Relāciju datu bāzē — datne, kur datu elementu atrašanās vieta rindās un kolonnās nosaka to savstarpējās attiecības.
2. Tekstapstrādē — izklājlapas, ko veido rindas un kolonnas un kas atvieglo matemātisku aprēķinu veikšanu.

1562 **table-oriented database management program**
**tabulorientēta datu bāzes pārvaldības programma**
**таблично ориентированная программа управления базой данных**

Datu bāzes pārvaldības programma, kas vaicājumoperācijas izpildes rezultātā displeja ekrānā izspīdina tabulu. Sk. arī *data retrieval, record-oriented database management system, Structural Query Language*.

1563 **Tabulator key**
**tabulēšanas taustiņš**
**клавиша табуляции**

Tastatūras taustiņš, kas parasti tiek izmantots tabulēšanas rakstzīmes ievadei. Dažās programmās tabulēšanas taustiņš tiek izmantots arī kādas ekrāna iezīmes pārvietošanai no vienas vietas uz otru. Piemēram, datu bāzu programmās tabulēšanas taustiņu bieži izmanto, lai pārvietotos pa ieraksta laukiem.

1564 **tag**
**birka, tags**
**тег, признак**

1. Viena vai vairākas rakstzīmes, kas piekārtotas datu kopai, lai to identificētu un sniegtu informāciju par kopu.
2. Kods, kas valodā *HTML* identificē kādu elementu (piem., noteiktu dokumenta daļu), lai globālā tīmekļa *WWW* pārlūkprogramma varētu nodrošināt tā izspīdināšanu displeja ekrānā. Šajā gadījumā tagus ievieto leņķiekavās (piem., <H1>).

1565 **tagged image file format (TIFF)**
**tagu attēlu datņu formāts, formāts TIFF**
**формат TIFF**

Firmu *Aldus* un *Microsoft* izstrādāts datņu standartformāts, ko izmanto grafisku attēlu glabāšanai datora atmiņā un kas nodrošina vienkrāsas un dažāda pelēkuma, kā arī 8 vai 24 bitu krāsu attēlu apstrādi. Praksē tiek izmantotas formāta *TIFF* versijas, kas atšķiras ar tajās iekļautajiem datu saspiešanas algoritmiem.

1566 **tape backup**
**lentdublējums**
**архивация на магнитной ленте**

Magnētiskās lentes izmantošana, lai glabātu cieto disku datņu rezerves kopijas.

1567 **target disk**
**mērķdisks**
**целевой диск**

Disks, kurā dati tiek ierakstīti vai pārkopēti no cita diska.

1568 **Tb**
Sk. *terabit.*

1569 **TB**
Sk. *terabyte.*

1570 **TCP**
Sk. *Transmission Control Protocol.*

1571 **tear-off menu**
**pārceļamā izvēlne**
**отрывное меню**
Izvēlne, ko var atraut no izvēlņu joslas un novietot jebkurā piemērotā displeja ekrāna vietā.

1572 **Telnet**
**protokols *Telnet***
**протокол *Telnet***
Tīkla *Internet* protokols, kas pieder pie protokolu *TCP/IP* saimes un ko izmanto attālai pieejai tīkla resursiem un terminālu emulēšanai. Šo protokolu bieži lieto, lai nodibinātu sakarus ar ziņojumdēļa sistēmu un lieldatoriem.

1573 **template**
**veidne**
**шаблон**
Šablons vai forma, ko teksta procesoram, izklājlapām vai citiem lietojumiem izmanto kā uzmetumu, aizpildot tukšās vietas ar vajadzīgo informāciju (piem., formulām, iezīmēm, skaitļiem, simboliem vai lokiem) un tādējādi atvieglojot atkārtotu uzdevumu risināšanu.

1574 **temporary font**
**pagaidu fonts**
**временный шрифт**
Fonts, kas pēc ielādēšanas lāzera printerī saglabājas tā atmiņā tik ilgi, kamēr to nomaina lietotājs vai programmatūra. Sk. arī *downloadable font, permanent font.*

1575 **tera-**
**tera-**
**тера-**
1. Prefikss, kas norāda uz kāda lieluma skaitliskās vērtības reizinājumu ar 1 biljonu ($10^{12}$).
2. Prefikss, kas datortehnikā norāda uz kāda lieluma skaitliskās vērtības reizinājumu ar $2^{40}$ (1 099 511 627 776).

1576 **terabit (Tb)**
**terabits**
**терабит**
Viens terabits ir $2^{40}$ jeb 1 099 511 627 776 biti. Sk. arī *tera-*.

1577 **terabyte (TB)**
**terabaits**
**терабайт**
Viens terabaits ir $2^{40}$ jeb 1 099 511 627 776 baiti. Sk. arī *tera-*.

1578 **terminal**
**terminālis**
**терминал**
Ievadizvades ierīce, kas pievienota datoram datu ievadīšanai un izvadīšanai. Terminālis parasti ir apgādāts ar tastatūru un displeju, kas lietotājam ļauj tieši sadarboties ar datoru. Sk. arī *dumb terminal, graphics terminal, smart terminal, video terminal.*

1579 **terminal emulation**
**termināļa emulācija;**
**termināļa emulēšana**
**эмуляция терминала**

Personālā datora izmantošana, lai imitētu kāda cita, parasti attāla, datora termināli. Šādu datora darba režīmu panāk, izmantojot speciālas programmas.

1580 **terminate-and-stay-resident program (TSR)**
***TSR* programma**
**программа *TSR***

Programma, kas tiek saglabāta atmiņā pat tad, ja tā attiecīgajā brīdī netiek izpildīta. Šo programmu var ātri palaist, lai izpildītu kādu specifisku uzdevumu.

1581 **test bed**
**testgultne**
**испытательный стенд**

Sistēmas vai tās atsevišķu komponentu testēšanas vide, kurā ir aparatūra, instrumentālie un valodas līdzekļi, imitācijas un cita nepieciešamā programmatūra, kas ļauj kontrolēt testēšanas starprezultātu apstrādi.

1582 **test data**
**testdati**
**тестовые данные**

Speciāli izveidota datu kopa, ko izmanto, lai pārbaudītu datora vai sistēmas darbības atbilstību paredzētajām prasībām. Kā testu var izmantot datus, kas iegūti sistēmas iepriekšējās darbības rezultātā vai arī kas speciāli radīti šīs pārbaudes veikšanai.

1583 **testability**
**testējamība**
**тестируемость**

Īpašība, kas raksturo sistēmas vai tās komponenta piemērotību testēšanas procedūras izpildei.

1584 **testing**
**testēšana**
**тестирование**

Programmatūras un aparatūras darbības pārbaude, izmantojot testdatus. Testēšanas mērķis var būt defekta atklāšana, tā atrašanās vietas lokalizēšana vai arī testējamā objekta dinamisko parametru (piem., ātrdarbības) noskaidrošana.

1585 **text**
**teksts**
**текст**

Cilvēkam uztverama informācija, kas sastāv no dabiskās vai mākslīgās valodas rakstzīmēm un teikumiem. Teksts ir viena no formām, kurā dators var uzglabāt un pārsūtīt informāciju. Parasti tekstu veido, izmantojot kodu *ASCII*.

1586 **text box**
**tekstlodziņš**
**текстовое окно**

Lodziņš displeja ekrānā, kas iezīmē laukumu, kurā lietotājs var ievadīt tekstu. Dažkārt šo lodziņu sauc arī par ievadlodziņu.

1587 **text redactor**
**teksta redaktors**
**текстовый редактор**

Datora programma, ko izmanto teksta datņu veidošanai un rediģēšanai. Atšķirībā no tekstprocesora teksta redaktors nenodrošina vārdu aplaušanas vai formatēšanas iespējas.

1588 **text file**
**teksta datne**
**текстовый файл**

Datne, kurā ir tikai koda *ASCII* rakstzīmes.

1589 **text mode**
**teksta režīms**
**текстовый режим**
Režīms, kurā displeja ekrāns tiek sadalīts rindās un kolonnās. Katrs rindas un kolonnas krustpunkts attēlo vienu rakstzīmi. Visi *IBM PC* tipa datoru videostandarti nodrošina teksta režīma izmantošanas iespējas. Dažkārt šo režīmu sauc arī par burtciparu vai rakstzīmju režīmu.

1590 **text-to-speech converter**
**tekstrunas pārveidotājs**
**преобразователь текста в речь**
Elektroniska ierīce, kas, izmantojot runas sintezatoru, pārveido datorā ievadīto tekstu vai kādu tā daļu runas ekvivalentā, tādējādi sagatavojot tā izvadi runas formā.

1591 **texture mapping**
**faktūrkartēšana**
**наложение текстуры**
Īpašs efekts datorgrafikā, ko panāk, izmantojot speciālus algoritmus, kas piešķir attēlam noteiktu virsmas faktūru (piem., marmora, ķieģeļu, akmens u.c.). Šo efektu var panākt arī elektroniski, aptinot vienu attēlu ap otru.

1592 **TFT screen**
Sk. *thin film transistor screen.*

1593 **thermal transfer printer**
**termopārneses printeris**
**принтер с термопереносом**
Bezuzsitiena printeris, kas veido krāsu attēlu, karsējot krāsotu vasku un pārnesot to uz papīra. Attēla veidošana notiek ar sakarsētām adatām, kas vispirms saskaras ar lenti, kas piesūcināta ar dažādās krāsās krāsotu vasku, izkausē to un pēc tam pārnes vasku uz parasta papīra.

1594 **thermal wax-transfer printer**
**vaska termopārneses printeris**
**принтер с термопереносом воска**
Sk. *thermal transfer printer*

1595 **thesaurus**
**tēzaurs**
**тезаурус**
Personālā datora atmiņā izveidota datne ar noteiktas vārdu kopas sinonīmiem un programma, ar kuras starpniecību var atrast vajadzīgā vārda sinonīmu šajā datnē.

1596 **thin film transistor screen**
**plānkārtiņu tranzistorekrāns**
**экран тонкопленочных транзисторов**
Augstas kvalitātes šķidro kristālu displejs, ko parasti lieto portatīvajos datoros un kura izgatavošanai izmanto plānkārtiņu tehnoloģiju. Sk. arī *active matrix display.*

1597 **thrashing**
**tukšdarbe**
**пробуксовка**
Virtuālās atmiņas sistēmas neefektīva darbība, kas ievērojami vairāk laika veltī lapošanai nekā lietojumprogrammas izpildei.

1598 **thread**
**pavediens**
**нить; цепочка выполняемых задач**
1. Laiksakritīgs process, kas ir kāda lielāka procesa vai programmas daļa. Šo

terminu parasti izmanto vairākuzdevumu operētājsistēmu aprakstos, kad vairākas programmas daļas (pavedieni) var tikt izpildīti vienlaicīgi.
2. Tematiski saistītu ziņojumu kopa, kas tiek nodota tīkla *USENET* intereškopai un kam tās dalībnieki izseko atbilstoši tematikai, nevis ziņojumu pienākšanas secībai.

1599 **throughput**
**caurlaidspēja**
**пропускная способность**
Darba daudzums, ko var veikt dators vai sakaru sistēma noteiktā laika posmā. Par caurlaidspējas mēru parasti izmanto datu apstrādes ātrumu datoru sistēmā vai datu pārsūtīšanas ātrumu sakaru sistēmā. Sk. arī *performance*.

1600 **thumbnail**
**sīktēls**
**маленькая страница**
Zīmējuma vai lappuses miniatūrs grafisks attēlojums, ko izmanto, lai ātri un vienkārši pārskatītu grafikas vai datorizdevniecības datņu saturu pirms to noformēšanas.

1601 **thumbwheel**
**īkšķrats**
**координатный манипулятор**
Rādītājierīces korpusā iemontēts ritenis, daļa no kura ir izvirzīta korpusa ārpusē. Griežot šo riteni ar pirkstu (parasti īkšķi), tiek vadīta kursora vai rādītājierīces rādītāja kustība displeja ekrānā. Īkšķratu izmanto trīsdimensiju kursorsvirās vai kursorbumbās, lai vadītu kursora vai rādītāja kustību trešajā dimensijā.

1602 **TIFF**
Sk. *tagged image file format*.

1603 **tilled windows**
**mozaīklogi**
**мозаичные окна**
Lietotāja saskarnē — divi vai vairāki logi, kas displeja ekrānā nepārklājas. Ja tiek atvērti papildu logi, pārējo logu izmēri vajadzības gadījumā automātiski tiek samazināti, lai uz ekrāna visi logi būtu redzami.

1604 **time division multiplexing**
**laikdales *multipleksēšana***
**временное мультиплексирование**
Multipleksēšanas paņēmiens, kas nodrošina vairāku sakaru kanālu izveidošanu vienā savienojošā kabelī (kopnē, ķēdē), sadalot kopējo pārraides laiku atsevišķos intervālos un piešķirot katram kanālam atbilstošu, periodiski atkārtojošos laika intervālu. Katram kanālam atbilstošajos laika intervālos pārraidītā informācija tiek apvienota uztvērējā. Sk. arī *frequency division multiplexing*.

1605 **title bar**
**virsrakstjosla**
**строка заголовка**
Taisnstūrveida josla loga augšējā daļā, kur norādīts loga vai tajā redzamā dokumenta nosaukums. Virsrakstjoslas krāsa norāda, vai logs ir aktīvs.

1606 **token passing**
**marķiera nodošana**
**передача маркера**
Pieejas metode vairāku lokālā tīkla staciju kopējai datu pārraides videi, izmantojot speciālu bitu kombināciju, ko sauc par marķieri. Marķieris tiek raidīts no vienas stacijas uz otru. Pēc tā saņemšanas (ja netiek ievērotas prioritātes) stacija drīkst monopolrežīmā izmantot

pārraides vidi. Marķiera nodošanas principi realizēti dažādos lokālo tīklu vides piecjas vadības protokolos. Sk. arī *Carrier Sense Multiple Access with Collision Detection, token-ring network.*

1607 **Token Ring network**
**tīkls *Token Ring***
**сеть *Token Ring***

Marķiergredzena tīkls, ko izstrādājusi firma *IBM*. Plaši izplatīti šīs izstrādnes divi varianti, kas atšķiras ar datu pārraides ātrumu (4 un 16 megabiti sekundē). Šim tīklam pievienojamo dažāda tipa datoru skaits var sasniegt 256.

1608 **token-ring network**
**marķiergredzena tīkls**
**сеть с маркерным кольцом**

Lokālā tīkla arhitektūra, kurā datu pārraidei starp mezgliem izmantots gredzentīkls. Plūsmas vadībai izmanto marķiera nodošanas mehānismu, kas novērš sadursmes starp vienlaicīgi raidošajām stacijām. Marķiergredzena tīkla darbības principi definēti standartā *IEEE 802.5* (*ISO 8802.5*). Sk. arī *IEEE 802 Standards.*

1609 **toner**
**toneris**
**тонер**

Elektriski uzlādēts pulveris vai tinte, ko izmanto elektrostatiskajos printeros un kopēšanas iekārtās. Attēls veidojas, tonerim pielīpot pretēji uzlādētai platei, veltnim vai papīram.

1610 **toner catridge**
**tonera kasetne**
**картридж с тонером**

Plastmasas kasetne, kurā iepilda toneri.

1611 **tool bar**
**rīkjosla**
**панель инструментов**

Taisnstūrveida josla loga augšējā daļā, kur izdalīti ar labi atšķiramām ikonām apzīmēti laukumiņi (pogas). Šīs ikonas pārstāv bieži lietojamas komandas.

1612 **toolbox**

Sk. *toolkit.*

1613 **toolkit**
**aprīkojums**
**набор инструментальных средств**

Rutīnu kopa, kas atvieglo gan programmas izstrādāšanu konkrētam lietojumam konkrētā vidē, gan arī šīs programmas atkļūdošanu.

1614 **touch screen**
**skārienekrāns**
**сенсорный экран**

Displeja ekrāns, kas pārklāts ar caurspīdīgu skārienjutīgu pārklājumu. Skārienekrāns pārveido lietotāja pirksta pieskārienu kādā noteiktā ekrāna vietā informācijā, kas tiek nodota programmatūrai. Tas dod iespēju skārienekrānu izmantot peles, gaismas zīmuļa vai tml. rādītājierīču aizstāšanai.

1615 **touch-sensitive display**
**skārienjutīgs displejs**
**сенсорный дисплей**

Sk. *touch screen.*

1616 **tower case**

Sk. *tower configuration.*

1617 **tower configuration**
**torņkonfigurācija**
**башенная конфигурация**

Personālā datora konfigurācija, kurā barošanas avots, mātesplate un ārējās atmiņas bloki izvietoti īpašā karkasā viens virs otra. Viena no šādas konfigurācijas priekšrocībām ir tā, ka trokšņojošus datora *funkcionālos* blokus var atvirzīt no darba vietas, novietojot šo korpusu, piem., uz grīdas blakus galdam.

1618 **traceability**
**trasējamība**
**трассируемость**
Pakāpe, kādā var nodibināt attiecības starp diviem vai vairākiem izstrādes procesā iegūtajiem produktiem, īpaši tādiem, kurus saista pakļautības attiecības (piem., priekšgājēja un pēcteča vai vedēja un sekotāja attiecības).

1619 **track**
**celiņš**
**дорожка**
Viens no vairākiem formatējot izveidotajiem disketes vai cietā diska koncentriskajiem riņķiem, kuros var ierakstīt un no kuriem var nolasīt datus.

1620 **trackball**
**kursorbumba**
**шар перемещения курсора**
Ievadierīce, ar kuru pārvieto kursoru displeja ekrānā, grozot šajā ierīcē iebūvētu lodveida manipulatoru. Atšķirībā no peles kursorbumba pati nav jāpārvieto un to parasti izmanto portatīvajos datoros. Sk. arī *joystick*.

1621 **tracks per inch**
**celiņi collā**
**число дорожек в дюйме**
Magnētiskā diska *informācijas pieraksta* blīvuma mērs, kas norada celiņu skaitu vienā diska radiusa collā.

1622 **trailer**
**noslēgums**
**завершитель, трейлер**
Identifikācijas vai vadības informācija, kas atrodas datnes vai ziņojuma beigās. Sk. arī *header*.

1623 **transaction**
**transakcija**
**транзакция**
1. Pakešapstrādes režīmā — darbs vai darba *solis*.
2. Sakaru sistēmās — mijiedarbība starp programmu lokālā sistēmā un programmu attālā sistēmā, lai iniciētu noteiktu darbību vai nodrošinātu vajadzīgā rezultāta iegūšanu.
3. Datu bāzēs — pieprasījums datu bāzes satura maiņai vai informācijas izguvei.

1624 **Transmission Control Protocol (TCP)**
**pārraides vadības protokols, protokols *TCP*, *TCP***
**протокол управления передачей данных, протокол *TCP***
Protokolu *TCP/IP* sistēmas sastāvdaļa, kas atbilst Atvērto sistēmu sadarbības bāzes etalonmodeļa transporta slāņa protokolam. Protokols *TCP* paredz iepriekšēju savienojuma nodibināšanu, un tajā ir tādi drošas datu pārraides un plūsmas vadības mehānismi kā slīdošais logs, taimauti, apstiprinājumi un atkārtotās pārraides. Sk. arī *User Datagram Protocol*.

1625 **Transmission Control Protocol/Internet Protocol (TCP/IP)**
**pārraides vadības protokols/intertīkla protokols, protokoli *TCP/IP*, *TCP/IP***
**протоколы *TCP/IP***

Protokolu sistēma, ko sākotnēji izstrādājusi ASV aizsardzības ministrijas Perspektīvo pētījumu aģentūra *DARPA*. Šajā sistēmā ietilpst vairāki desmiti protokolu (*TCP*, *IP*, *UDP*, *FTP* u.c.), kas kļuvuši par tīkla *Internet* bāzi. Protokola *TCP/IP* detalizēts apraksts atrodams *RFC* dokumentos.

1626 **trap**
**slazds**
**ловушка**

Speciāla ierīce, programmatūra vai aparatūra, ar kuru tiek fiksētas noteiktas situācijas (pārtraukumi, klūdas, instrukciju izpildes sākums u.c.), kas rodas programmas izpildes vai atklūdošanas gaitā.

1627 **trapping**
**slazdošana**
**организация ловушек**

Noteiktas darbības veikšana, lai pārtrauktu darbojošos programmu, pārbaudītu un analizētu tās stāvokli, un, ja nepieciešams, nodrošinātu tās atkopšanu.

1628 **tree**
**koks**
**дерево**

Hierarhiska datu struktūra, kuras visi elementi, izņemot saknes elementu, satur atsauci uz vienu vai vairākiem zemāka līmeņa datu struktūras elementiem un vienu augstāka līmeņa datu struktūras elementu. Sk. arī *directory tree*.

1629 **TrueType**
**tehnoloģija *TrueType***
**технология *TrueType***

Firmu *Apple* un *Microsoft* fontu veidošanas tehnoloģija, kas nodrošina mērogojamu kontūrfontu attēlošanu displeja ekrānā un nodošanu printeriem firmas *Apple* operētājsistēmu un operētājsistēmas *Microsoft Windosws* vidēs. *TrueType* fontu izdruka ir identiska to attēlojumam displeja ekrānā. Katrā kontūrfontā ir algoritms, kas nodrošina tā bitkartēta mērogojama attēla izveidi kā ekrānam, tā arī printerim un nav nepieciešamas papildu utilītprogrammas vai interpretatori kā, piem., *PostScript* fontiem.

1630 **turnkey system**
**darbgatava sistēma**
**система, готовая к непосредственному использованию**

Aparatūras vai programmatūras sistēma, kas tūlīt pēc tās uzstādīšanas ir gatava izmantošanai. Šādas sistēmas parasti tiek izstrādātas kādam konkrētam lietojumam.

1631 **turtle**
**bruņrupucis**
**черепаха**

Ierīce grafisku attēlu zīmēšanai, ko parasti vada, izmantojot valodas *Logo* instrukcijas. Izšķir divus šādu ierīču veidus. Ekrāna bruņrupucis ir trīsstūrveida rādītājs, kas darbojas līdzīgi gaismas zīmulim un kas, to pārvietojot pa displeja ekrānu, zīmē kādu grafisku attēlu. Grīdas bruņrupucis ir elektromehāniska ierīce, kas attēlus zīmē uz papīra, pārvietojoties pa līdzenu virsmu (piem., grīdu).

1632 **turtle graphics**
**bruņrupučgrafika**
**черепашья графика**

Grafisku attēlu veidošana, izmantojot bruņrupuci un virkni relatīvi vienkāršu instrukciju (parasti valodā *Logo*), kas nosaka tā kustības virzienu.

1633 **twisted pair**
**vītais pāris**
**скрученная пара [проводов]**
Divi savīti izolēti vadi, kas samazina indukciju un ar to saistīto elektrisko signālu interferenci. Vītais pāris izveido samērā lētu informācijas pārraides vidi. To bieži lieto neliela ātruma lokālajos tīklos un telefonijā.

1634 **type style**
**burtstils**
**тип шрифта**
Noteikta burtveidola rakstzīmju izpildījuma veids, piem., treknraksts, slīpraksts.

1635 **typeface**
**burtveidols**
**гарнитура шрифта**
Speciāla drukāto rakstzīmju izveide, ko raksturo to slīpums un līniju biezums. Burtveidols atšķiras no fonta, ar ko parasti saprot īpaša burtveidola noteiktu lielumu, kā arī no burtveidolu saimes, kas ir radniecīgu burtveidolu grupa. Sk. arī *bold*, *boldface*, *bold italic*, *italic*.

1636 **typesetter**
**rakstlicis**
**наборное устройство**
Sk. *phototypsetter, image setter.*

## U

1637 **UDP**
Sk. *User Datagram Protocol.*

1638 **Ultimedia**
**koncepcija *Ultimedia***
**концепция *Ultimedia***
Firmas *IBM* izstrādāta multivides koncepcija, kas dod iespēju apvienot audio-, video- un tekstuālās informācijas apstrādi un nosaka, kāda aparatūra nepieciešama šim nolūkam.

1639 **underline**
**pasvītrojums**
**подчеркивание**
Atsevišķu teksta daļas izcelšanas paņēmiens, nodrukājot zem tās līniju. Dažas tekstu apstrādes programmas dod iespēju pasvītrojumu izdarīt kā ar vienkāršu līniju, tā arī ar dubultlīniju vai punktlīniju.

1640 **undo**
**atsaukt**
**отменить**
Atgriezties iepriekšējā stāvoklī, likvidējot pēdējo izdarīto darbību, piem., atsaucot pēdējo dzēšanu, var atjaunot iepriekšējo dokumenta tekstu.

1641 **Uniform Resource Locator (URL)**
**vienots resursu vietrādis, vietrādis *URL*, *URL***
**унифицированный указатель ресурсов**
Adrese, kas pārlūkprogrammā norāda, kur var atrast kādu konkrētu tīkla *Internet* resursu. Vienveida resursu vietrādim varētu būt, piem., šāda struktūra: *http:// www edzi.lza.lv. /history.htm* Vietrāža pirmā daļa (līdz dubultajai slīpsvītrai) norāda pieejas metodi (šajā piemērā — hiperteksta transporta protokolu). Teksts starp dubulto slīpsvītru un vienkāršo slīpsvītru norāda serveri, bet teksts aiz vienkāršās slīpsvītras — direktoriju vai datni.

1642 **uninterruptible power supply (UPS)**
**nepārtrauktā barošana**
**источник бесперебойного питания**

Datoru sistēmas nodrošinājums ar dublējošu barošanas avotu gadījumam, ja parastais barošanas avots tiek atslēgts vai tā spriegums samazinās līdz nepieļaujamam līmenim. Vienkāršākais nepārtrauktās barošanas nodrošinājuma veids ir bojātā barošanas avota aizstāšana ar bateriju, kas nodrošina datora funkcionēšanu tik ilgu laiku (dažas minūtes), lai tā darbību varētu normāli pārtraukt. Ja nepieciešams saglabāt datoru darba stāvoklī ilgāku laiku (dažas dienas), tad kā dublējošu barošanas avotu izmanto speciālu ģeneratoru. Barošanas avotus, kas nodrošina šādu režīmu, sauc par nepārtrauktās barošanas avotiem.

1643 **unit**
**vienība; bloks; mezgls**
**единица; блок; узел**

1. Patstāvīga vienota veseluma daļa.
2. Loģiski patstāvīga programmas daļa.
3. Ierīce, kas veic kādu noteiktu funkciju kopumu. Sk. arī *block, central processing unit, node, memory management unit, system unit, vertical format unit.*

1644 **UNIX**
**operētājsistēma *UNIX***
**операционная система *UNIX***

Vairāklietotāju un vairākuzdevumu operētājsistēma, ko 70. gadu sākumā izstrādāja firmas *AT&T* pētniecības centrā *Bell Laboratories*. Operētājsistēma *UNIX* ir elastīga operētājsistēma, ko var adaptēt dažādiem lietotājiem un izmantot gan lieldatoros, gan personālajos datoros. Sk. arī *Advanced Interactive eXecutive.*

1645 **updating**
**atjaunināšana**
**обновление**

Datnes vai datu bāzes satura izmaiņa, kas atkarībā no sistēmas konkrētā stāvokļa var ietvert sevī jaunu datu ierakstīšanu, dzēšanu vai atsevišķu fragmentu aizstāšanu ar citiem.

1646 **upgrade**
**jauninājums**
**усовершенствование**

1. Datoru sistēmas rekonfigurācija tās veiktspējas uzlabošanai.
2. Programmatūras *modificēšana* vai jaunu versiju ieviešana, lai uzlabotu tās funkcionēšanu.

1647 **UPS**

Sk. *uninterruptible power supply.*

1648 **upward compatible**
**augšupsaderīgs**
**совместимость снизу вверх**

Datoru un programmatūras īpašība, kas norāda, ka dators var izpildīt visas iepriekšējā modeļa funkcijas un ka programmatūrā ietvertas iepriekšējo šīs programmatūras versiju iespējas.

1649 **URL**

Sk. *Uniform Resource Locator.*

1650 **usability**
**lietojamība**
**используемость**

Sistēmas īpašība, kas raksturo, cik viegli lietotājs var apgūt tās izmantošanu, sagatavot tai ieejas datus un interpretēt tās izejas datus.

1651 **USENET (USEr NETwork)**
**tīkls *USENET***
**сеть *USENET***

Starptautisks nekomerciāls datoru tīkls, kas apvieno operētājsistēmas *UNIX* vidē strādājošas datoru sistēmas, kuras ar tīkla *Internet* starpniecību tiek saistītas ar citiem datoru tīkliem. Parasti tīklu *USENET* izmanto kā dalītu ziņojumdēļa sistēmu, ar kuras palīdzību dažādas intereškopas apmainās ar informāciju.

1652 **user**
**lietotājs**
**пользователь**

Persona, kas izmanto datoru un tā lietojumprogrammas savu uzdevumu risināšanai.

1653 **user account**
**lietotāja konts**
**конт пользователя**

Drošības mehānisms, ko izmanto, lai kontrolētu pieeju datoru sistēmai vai tīklam. Lietotāja kontu nosaka un pārrauga sistēmas administrators. Lietotāja kontā ietilpst informācija par lietotāja paroli, tā tiesībām un, iespējams, cita informācija, kas ļauj identificēt lietotāju.

1654 **User Datagram Protocol (UDP)**
**lietotāja datogrammu protokols, protokols *UDP*, *UDP***
**протокол передачи пользовательских датаграмм, протокол *UDP***

Protokola *TCP/IP* sastāvdaļa, kas atbilst atvērto sistēmu sadarbības bāzes etalonmodeļa transporta slānim. Šī protokola galvenā funkcija ir lietotāju datogrammu pārraides vadība. Atšķirībā no protokola *TCP*, protokols *UDP* paredz bezsavienojuma režīma izmantošanu un līdz ar to negarantē drošu datu pārraidi.

1655 **user interface**
**lietotāja saskarne**
**интерфейс пользователя**

Visu programmā vai datorā paredzēto līdzekļu kopums, kas nosaka, kā lietotājs var sadarboties ar datoru. Lietotāja saskarni veido, piem., izvēļņu kombinācijas, ekrāna plānojums, tastatūras komandas, komandvaloda, kā arī dažādas ievadizvades ierīces. Sk. arī *command-driven program*, *graphical user interface*, *menu-driven program*.

1656 **user object**
**lietotāja objekts**
**объект пользователя**

Informācija, ko lietotājs ievada lietojumprogrammā. Šo terminu parasti izmanto firmas *IBM* izstrādātajā kopējā lietotāju pieejas arhitektūrā. Sk. arī *application object*.

1657 **user option**
**lietotāja opcija**
**опция пользователя**

Lietojumprogrammas un lietotāja mijiedarbības izvēle, ko lietotājam nodrošina lietojumprogramma tās darbības laikā. Šo terminu parasti izmanto firmas *IBM* izstrādātajā kopējā lietotāju pieejas arhitektūrā. Sk. arī *application option*.

1658 **user-friendly**
**lietotājdraudzīgs**
**дружественный**

Termins apzīmē datu apstrādes sistēmas, datorus, ierīces vai programmas, ko viegli apgūt un izmantot un ar ko strādājot lietotājam nevajag detalizēti izstudēt dažādus papildavotus (darbības aprakstus, rokasgrāmatas u.c.).

1659 **username**
**lietotāja vārds**
**имя пользователя**
Viena no datoru sistēmu un tīkla aizsardzības formām, kas (parasti, kopā ar paroli) ļauj autentificēt lietotāju.

1660 **utility program**
**utilītprogramma**
**сервисная программа**
Nelieļa programma vai šādu programmu kopa, kas nodrošina papildpakalpojumus, ko nesniedz oprētājsistēma. Personālajiem datoriem parasti ir daudz atkārtoti izpildāmu uzdevumu (cieto disku defragmentēšana, reti lietojamu datņu saspiešana, datņu atkopšana u.c.), ko veic utilītprogrammas.

## V

1661 **vaccine**
**vakcīna**
**вакцина**
Programma, kas nodrošina datora aizsardzību pret vīrusiem. Vakcīna veic sistēmas pārbaudi, lai atklātu un likvidētu tajā esošos vīrusus.

1662 **validation**
**validēšana; validācija**
**проверка достоверности**
Sistēmas pārbaude, ko veic tās izstrādāšanas beigu posmā, lai pārliecinātos, ka izstrādātās sistēmas funkcionēšana atbilst iepriekš formulētajām prasībām. Sk. arī *verification*.

1663 **value-added reseller (VAR)**
**pievienotās vērtības tālākpārdevējs**
**реселлер, вносящий добавленную стоимость**
Firma, kas palielina datoru sistēmas vērtību, uzlabojot tās dokumentāciju, izmantošanas iespējas, sniegto pakalpojumu loku, sistēmas integritāti un citas ar tās lietošanu saistītās īpašības. Pēc šo uzlabojumu īstenošanas firma sistēmu pārdod tālāk, nosakot tās cenu atbilstoši veiktajiem uzlabojumiem.

1664 **VAR**
Sk. *value-added reseller*.

1665 **vector display**
**vektordisplejs**
**векторный дисплей**
Katodstaru lampas displejs, kas ļauj elektronu staram brīvi sekot signālam, kurš nosaka tā koordinātas. Sk. arī *raster display*.

1666 **vector graphics**
**vektorgrafika**
**векторная графика**
Visplašāk pazīstamais grafisku datu attēlošanas veids, kuru izmantojot datora izvadierīce vektorus attēlo, pa punktiem zīmējot tiem atbilstošas taisnes un līknes.

1667 **vector processor**
**vektoru procesors**
**векторный процессор**
Procesors, kas vienlaicīgi operē ar vienu matricas rindu vai kolonnu. Sk. arī *array processor*.

1668 **vector-to-raster conversion**
**vektorrastra konversija**
**векторно-растровое преобразование**
Vektorgrafikas pārveidošana bitkartētos attēlos, kas veidoti no punktiem vai pikseļiem. Sk. arī *bit-mapped graphics*, *raster-to-vector conversion*.

1669 **Ventura Publisher datorizdevniecības programma *Ventura Publisher*, *Ventura Publisher* издательская программа *Ventura Publisher***

Datorizdevniecības programma, ko plaši izmanto rakstnieki un redaktori darbam ar liela apjoma dokumentiem un dokumentu bibliotēkām. Šī programma paredzēta *Macintosh* saimes un firmas *IBM* vai ar tiem saderīgajiem datoriem.

1670 **verification verificēšana; verifikācija верификация**

1. Programmas korektuma formāla pierādīšana.
2. Sistēmas pārbaude, ko parasti veic tās izstrādāšanas gaitā, lai pārliecinātos, vai izstrādāšanas procesā aplūkojamā posma rezultāti atbilst tā sākumā definētajiem noteikumiem un prasībām. Sk. arī *validation*.

1671 **vertical format unit (VFU) vertikālā formāta bloks, bloks *VFU* блок вертикального формата**

Printera vadības sistēmas daļa, kas nosaka drukājamā dokumenta veidošanu vertikālajā virzienā (piem., atstarpju vertikālos parametrus, lappušu garumu).

1672 **vertical justification vertikālā līdzināšana вертикальное выравнивание**

Teksta rindiņu atstarpju lieluma regulēšana, lai nodrošinātu kādas teksta daļas ievietošanu veidojamajā lappusē.

1673 **VESA**

Sk. *Video Electronics Standards Association*.

1674 **VESA local bus**

Sk. *VL-bus*.

1675 **VFU**

Sk. *vertical format unit*.

1676 **VGA**

Sk. *Video Graphics Array*.

1677 **video adapter videoadapteris видео адаптер**

Elektroniska ierīce, ar kuru ģenerē videosignālus, kas pa kabeli tiek nosūtīti displejam. Šo ierīci dažkārt sauc arī par videokontrolleri. Videoadapteris konstruktīvi var būt iebūvēts datorā vai arī izveidots kā termināļa sastāvdaļa. Teksta vai grafikas attēla kvalitāte ir atkarīga no izmantotā videoadaptera un monitora, kuru kopīga darbība nodrošina izvēlētā videostandarta prasību izpildi.

1678 **video board videoplate видеоплата**

Paplašināšanas plate, kas tiek pievienota personālā datora spraudņiem un kas ģenerē tekstu vai grafikas attēlus monitora ekrānā. Videoplati dažkārt sauc arī par videoadapteri, videokontrolleri vai videokarti.

1679 **video buffer videobuferis видео буфер**

Atmiņa videoadapterī, kas paredzēta displeja ekrānā rādāmo datu glabāšanai. Teksta apstrādes režīmā šie dati tiek glabāti kodā *ASCII*. Grafikas režīmā katru pikseli nosaka viens vai vairāki biti. Bitu skaits nosaka displeja ekrānā vienlaicīgi redzamo krāsu skaitu.

1680 **video capture board**
**videotvērējplate**
**плата захвата видео**
Specializēta ātrdarbīga karte, kas video-attēlu pārveido ciparsignāla formā un ļauj ierakstīt to datora atmiņā turpmākajai apstrādei. Sk. arī *capture*.

1681 **video card**
**videokarte**
**видеокарта**
Sk. *video board*.

1682 **video conferencing**
**videokonference**
**видеоконференция**
Datora tīkla pakalpojumu kopums, kas nodrošina attālu lietotāju interaktīvus sakarus reāllaikā, izmantojot video- un audioinformāciju. Organizējot videokonferenci, videoinformācijas apmaiņas nodrošināšanai izmanto īpašu platjoslas datu pārraides tīklu.

1683 **video digitizer**
**videociparotājs**
**цифровой видеопреобразователь**
Sk. *frame grabber*.

1684 **video display**
**videodisplejs**
**видеодисплей**
Sk. *display*.

1685 **video editing**
**videorediģēšana**
**видеоредактирование**
Videokadru secības rediģēšanas metode, kas paredz videokadru pārveidošanu ciparu formā un uzglabāšanu datora atmiņā, lai, izgriežot un pārvietojot šos kadrus, sakārtotu tos jebkurā kārtībā pirms to izvadīšanas no datora galīgā secībā.

1686 **Video Electronics Standards Association (VESA)**
**Videoelektronikas standartu asociācija, asociācija *VESA*, *VESA***
**Ассоциация стандартов видеоэлектроники, ассоциация *VESA***
Personālo datoru videoadapteru un monitoru ražotāju organizācija, kas nodarbojas ar videostandartu izstrādāšanu un uzlabošanu. *VESA* piedalījusies video kontrolleru, monitoru un standartu (t.sk. *VGA*, *Super VGA*, *XGA*) izstrādāšanā.

1687 **Video Graphics Array (VGA)**
**videostandarts *VGA*;**
**videoadapteris *VGA*, adapteris *VGA***
**видеостандарт *VGA*;**
**видеоадаптер *VGA*;**
**адаптер *VGA***
1. Videostandarts, kas izstrādāts *IBM PS/2* datoriem un papildina sava priekšgājēja — videostandarta *EGA* — iespējas. Teksta režīmā šis standarts 16 krāsām paredz izšķirtspēju 720×400, bet grafikas režīmā 16 krāsām — 640×480 un 256 krāsām — 320×200.
2. Videoadapteris, kas atbilst videostandartam *VGA* un, izmantojot piemērotu monitoru (parasti analogo), nodrošina tā prasību izpildi.

1688 **video look-up table**
**videopārlūktabula**
**таблица преобразования видеосигнала**
Sk. *color look-up table*.

1689 **video mode**
**videorežīms**
**видеорежим**
Režīms, kas nodrošina *IBM PC* vai *IBM PS/2* datoru iespēju attēlot displeja ekrānā grafiku un kas atkarīgs no datorā instalētās izvērses plates un tam pievienotā monitora. Neatkarīgi no tā, kāds video adapteris ir instalēts, visi šie datori var strādāt ar tekstiem. Sk. arī *graphics mode*, *text mode*.

1690 **video monitor**
**videomonitors**
**видеомонитор**
Sk. *monitor*.

1691 **video RAM (VRAM)**
**brīvpieejas videoatmiņa**
**видеопамять**
Speciāli projektēta videoadaptera atmiņa, ko izmanto displeja ekrāna attēlu saglabāšanai. Dati no brīvpieejas videoatmiņas ar programmu starpniecību tiek translēti monitoram vajadzīgajā formātā. Parasti izmanto divu pieslēgvietu brīvpieejas videoatmiņu, kas ļauj vienlaicīgi nolasīt un ierakstīt datus.

1692 **video standard**
**videostandarts**
**видеостандарт**
Norunu kopums, kas nosaka displeja izšķirtspēju un krāsas. Galvenie videostandartu izstrādātāji ir firma *IBM*, *Hercules Computer Technology Inc.* un asociācija *VESA*. Videostandartu nodrošina kā izmantotais monitors, tā arī videoadapteris. Monitoram jābūt spējīgam veidot attēlu ar videostandartā noteikto izšķirtspēju un krāsām, bet videoadapterim — pārraidīt attiecīgus signālus monitoram. Biežāk tiek izmantoti videostandarti *CGA*, *EGA*, *DGA*, *MDA*, *MCGA*, *SVGA*, *VGA*, *XGA* kā arī *Hercules Graphics*.
*Piezīmes*:
1. Tā kā grafikas sistēmas nepārtraukti attīstās, būtu vēlama vārdnīcā norādīto videostandartu skaitlisko raksturojumu precizēšana, iepazīstoties ar jebkuru jaunu sistēmu.
2. Vārdnīcā ar terminu "videoadapteris" saprot atbilstošā videostandarta implementāciju aparatūrā un/vai programmatūrā neatkarīgi no tās konstruktīvajām īpatnībām.

1693 **video terminal**
**videoterminālis**
**видеотерминал**
Terminālis, kas datu ievadīšanai parasti izmanto tastatūru un datu izvadīšanai monitoru ar atbilstošu videoadapteri. Videoterminālis dažkārt veic arī zināmas izvadāmo datu apstrādes operācijas.

1694 **video window**
**videologs**
**видеоокно**
Logs, kurā neatkarīgi no pārējās displeja ekrānā attēlotās informācijas izspīdina kustīgus videoattēlus.

1695 **videodisc**
**videodisks**
**видеодиск**
Optiskais disks, kas paredzēts tikai ierakstīto datu nolasīšanai un ko izmanto video- un ar to saistītās audioinformācijas uzglabāšanai un reproducēšanai.

1696 **videotex**
**videotekss**
**видеотекс**
Interaktīva teksta un grafikas attēlu pār-

raidīšanas sistēma, kas nodrošina datu izguvi no datu bāzēm un to pārsūtīšanu lietotāja terminālim pa telefona līnijām. Atšķirībā no teleteksta, kur lappuses tiek pārsūtītas automātiski viena pēc otras, videoteksa lietotājs, izmantojot telefona līniju un modemu, izsauc vienu lappusi ar viņam nepieciešamo informāciju.

1697 **view**
**skatīt; skatījums**
**просматривать; область просмотра**

1. Pārlūkot uz displeja ekrāna attēlotos datus.
2. Loģiska tabula (apakšshēma) relāciju datu bāzu pārvaldības sistēmās, kurā īslaicīgi var apvienot vienu vai vairākas datnes, lai tās varētu attēlot displeja ekrānā, drukāt un no tām saņemt kādu informāciju. Sk. arī *schema*.

1698 **viewer**
**skatītājs**
**средство просмотра**

Palīgprogramma, kas lietotājam ļauj redzēt attēlu un dokumentu datņu saturu, neizmantojot programmu, ar kuru šīs datnes izveidotas.

1699 **virtual desktop**
**virtuālā darbvirsma**
**виртуальный рабочий стол**

Par displeja ekrāna fiziskajām robežām lielāka darbvirsma, kurā ir, piem., teksts, attēli, logi. Izmantojot virtuālo darbvirsmu, ekrānā redzams logs, kurā var tikt ritinātas dažādas virtuālās darbvirsmas daļas.

1700 **virtual disk**
**virtuālais disks**
**виртуальный диск**

Sk. *RAM disk*.

1701 **virtual image**
**virtuālais attēls**
**виртуальное изображение**

Datora atmiņā ievadīts grafisks attēls, kura izmēri ļauj to aplūkot displeja ekrānā tikai pa daļām. Sk. arī *virtual desktop*.

1702 **virtual machine**
**virtuālā mašīna**
**виртуальная машина**

1. Virtuāla datu apstrādes sistēma, kas šķietami nodota katra atsevišķa lietotāja rīcībā, bet kuras darbība tiek nodrošināta, virtuālo mašīnu lietotājiem kopīgi izmantojot reālās datu apstrādes sistēmas resursus.
2. Firmas *Intel* mikroprocesoriem (sākot ar *Intel 80386*) — procesora aparatūrā izveidots aizsargāts atmiņas apgabals. Katra virtuālā mašīna neatkarīgi no citas virtuālās mašīnas var izpildīt savas programmas. Virtuālās mašīnas var organizēt pieeju tastatūrai, printerim un citām iekārtām. Katru virtuālo mašīnu vada tai piemērota operētājsistēma.

1703 **virtual memory**
**virtuālā atmiņa**
**виртуальная память**

Datora brīvpieejas atmiņas šķietama palielināšana, šim nolūkam izmantojot daļu cietā diska.

1704 **virtual reality**
**virtuālā realitāte**
**виртуальная реальность**

Datorā veidota mākslīga trīsdimensiju vide, kas rada realitātes ilūziju. Lai panāktu šo ilūziju, tiek izmantoti speciāli stereoskopiska efekta displeji (brilles) un palīgierīces manipulācijām ar virtuālās realitātes objektiem. Sk. arī *cyberspace*.

1705 **virus**
**vīruss**
**вирус**

Programma, kas patvaļīgi pievienojas citām datora programmām un to darba laikā veic dažādas nevēlamas darbības: bojā datnes, katalogus un skaitļošanas rezultātus, dzēš vai piesārņo atmiņu kā arī citādi traucē datora darbību. Sk. arī *vaccine*.

1706 **Visual Basic programmēšanas valoda**
***Visual Basic*, *Visual Basic***
**язык программирования *Visual Basic***

Augsta līmeņa programmēšanas valoda lietojumprogrammu izstrādei *Microsoft Windows* vidē. Izmantojot *Visual Basic*, izveidotas ērtas lietotāju saskarnes, kas nodrošina dažādu objektu, piem., izvēlņu, dialoglodziņu u.c. elementu, atlasi no displeja ekrānā redzamā izstrādātāja aprīkojuma un to izvietošanu konkrētajā lietojumprogrammā. *Visual Basic* programmatūra nodrošina objekta sasaisti un iegulti, kā arī īsteno protokola "Dinamiskā datu apmaiņa" procedūras.

1707 **visual page**
**vizuālā lappuse**
**отображаемая страница**

Viena no ekrānbuferī glabājamajām lappusēm ar krāsu videoadapteri aprīkotajos personālajos datoros. Sk. arī *active page*.

1708 **visualization**
**vizualizēšana**
**визуализация**

Cipardatu pārveidošana grafiskā formā, lai lietotājam atvieglotu attēlu atpazīšanu. Attēlu identifikācija ciparu formā ir apgrūtināta.

1709 **VL-bus**
**kopne VL**
**шина VL**

Asociācijas *VESA* izstrādāta lokālās kopnes arhitektūra. Atbilstoši šai arhitektūrai izveidotās lokālās kopnes paplašināšanas slotiem var tikt pieslēgti trīs dažādi vedēji (piem., videoadapteri, cietā diska adapteri un tīkla adapteri), kas bez centrālā procesora līdzdalības ļauj ātri nosūtīt datus atmiņai. Kopni *VL* ar arhitektūras *EISA*, *ISA* vai *MCA* kopnēm savieno, izmantojot kopnes tiltus.

1710 **voice input**
**runas ievade**
**речевой ввод**

Informācijas (t.sk. instrukciju) ievadīšana datorā, izmantojot runas pazīšanas tehnoloģiju. Sk. arī *speech recognition*.

1711 **voice output**
**runas izvade**
**речевой вывод**

Sk. *speech synthesis*.

1712 **voice recognition**

Sk. *speech recognition*.

1713 **voice response**
**runas atbilde**
**речевой ответ**

Datora ģenerētas runas izvade. Atbildot uz lietotāja komandām, dators nevis izvada tekstu displeja ekrānā, bet sniedz atbildi balsī.

1714 **voice synthesizer**

Sk. *speech synthesizer*.

1715 **volatile memory**
**enerģoatkarīga atmiņa**
**энергозависимая память**

Atmiņa, kuras saturs zūd, ja tai pārtrauc pievadīt barošanas spriegumu. Enerģoatkarīga ir, piem., brīvpieejas atmiņa. Sk. arī *nonvolatile memory*.

**1716 VRAM**
Sk. *video RAM*.


**1717 wallpaper**
**tapete**
**обои**
Attēls vai šablons, ko lietotāja grafiskajā saskarnē izmanto kā loga fonu.

**1718 WAN**
Sk. *wide area network*.

**1719 wand**
**zizlis**
**зонд жезлового типа**
Rokā turama ierīce, kas apgādāta ar optisko skeneri un ir izmantojama rokraksta, mašīnraksta, iespiesta teksta, rakstzīmju, ka arī svītrkoda nolasīšanai un ievadīšanai datorā vai citā datu apstrādes iekārtā.

**1720 WAR file**
Sk. *WAVE file*.

**1721 warm boot**
**siltā sāknēšana**
**горячая самозагрузка**
Sistēmas programmatūras atkārtota ielāde vai inicializācija, kas notiek pēc tam, kad sistēmai ir pieslēgts barošanas spriegums un tā ir darba stāvoklī. Sk. arī *cold boot*.

**1722 warning message**
**brīdinājuma ziņojums**
**предупреждающее сообщение**
Ziņojums, kas informē lietotāju, ka pieprasītā darbība tiek pārtraukta, jo noticis kas neparedzēts. Lietotājs var turpināt darbību vai atlikt tās izpildi, kā arī griezties pēc palīdzības. Sk. arī *action message*, *information message*.

**1723 WAVE file**
***WAVE* datne**
***WAVE* файл**
Operētājsistēmā *Microsoft Windows* izmantotā metode analogsignālu uzglabāšanai ciparu formā. Šo datņu paplašinājums ir *.WAV*.

**1724 WAV file**
Sk. *WAVE file*.

**1725 Web browser**
Sk. *browser*.

**1726 Web server**
**tīmekļa serveris, *Web* serveris**
**сервер Web**
Globālā tīmekļa *WWW* programma, kas atbilstoši protokola *HTTP* prasībām pieņem kadrētus informācijas pieprasījumus, apstrādā tos un nosūta lietotājam pieprasīto dokumentu.

**1727 webmaster**
**tīmekļa pārzinis**
**управляющий паутиной**
Persona, kas ir atbildīga par kāda tīmekļa *WWW* apgabala (zonas) administrēšanu.

1728 **Whetstone**
**etalonprogramma *Whetstone***
**эталонная тестовая программа *Whetstone***
Etalonprogramma, ko izmanto, lai noteiktu un salīdzinātu dažādu datoru veiktspēju, tiem izpildot aritmētiskās operācijas ar peldošo komatu (punktu). Veiktspēja tiek raksturota ar šīs programmas izpildes reižu skaitu sekundē. Sk. arī *Dhrystone*.

1729 **white pages**
**baltās lapas**
**белые страницы**
Tīkla *Internet* izziņu dienests, kurā ir informācija par tā lietotājiem — elektroniskā pasta adreses, telefona numuri, pasta adreses u.c., kas atvieglo šīs informācijas izguvi. Termins veidojies, izmantojot analoģiju ar telefonu grāmatu, kur šāda rakstura informācija parasti tiek iekļauta t.s. baltajās lapās.

1730 **wide area network (WAN)**
**teritoriālais tīkls**
**территориальная сеть**
Datoru tīkls, kas savieno attālus lietotājus, kuri var atrasties citās pilsētās vai valstīs un kuri parasti izmanto vispārējās lietošanas vai speciālus sakaru līdzekļus. Sk. arī *local area network*, *metropolitan area network*.

1731 **widow [line]**
**atraitņrinduļa**
**изолированная строка**
Tekstapstrādē — pēdējā paragrāfa rinduļa, kas parādās viena pati nākošās lappuses augšā. Vairums teksta apstrādes programmu un izklājprogrammu nepieļauj šādu atsevišķu rinduļu parādīšanos.

1732 **wildcard character**
**aizstājējzīme**
**безразличный символ**
Tastatūras rakstzīme, ko izmanto vienas vai vairāku rakstzīmju vietā. Operētājsistēmā *MS-DOS* ar jautājuma zīmi (?) var aizstāt jebkuru vienu rakstzīmi, ar zvaigznīti (*) var aizstāt jebkuru skaitu rakstzīmju.

1733 **WIMP interface**
Sk. *Windows*, *Icons*, *Mouse*, *Pointers interface*.

1734 **window**
**logs**
**окно**
Displeja ekrāna laukums, kurā parādās informācija, kas attiecas uz darba procesā izmantojamajiem objektiem (programmām, dokumentiem, ziņojumiem u.c.). Informācija par dažādiem objektiem var tikt parādīta vienlaicīgi vairākos logos. Izšķir aktīvos un neaktīvos logus. Sk. arī *windowing environment*.

1735 **window title**
**loga virsraksts**
**заголовок окна**
Laukums virsrakstjoslā, kurā atkarībā no loga tipa var būt tajā attēlotās programmas, datnes, dokumenta vai cita objekta nosaukums, ar ko identificē logu. Aktīvajā logā šis laukums tiek izcelts, lai pievērstu tam lietotāja uzmanību.

1736 **windowing**
**logošana**
**кадрирование**
Paņēmienu un metožu kopums displeja ekrāna sadalīšanai divās vai vairākās daļās, kurās vienlaicīgi var tikt attēlota dažāda informācija, ko iespējams apstrādāt gan atsevišķi, gan kopīgi.

1737 **windowing environment**
**logošanas apkārtne**
**оконная среда**
Jebkura operētājsistēma vai operētājsistēmas paplašinājums, kas ļauj lietotājam displeja ekrānā strādāt ar dažāda izmēra logiem, kā arī izmantot grafiskās lietotāja saskarnes iespējas.

1738 **Windows**
Sk. *Microsoft Windows*.

1739 **Windows, Icons, Mouse, Pointers interface (WIMP interface)**
***WIMP* saskarne**
**интерфейс *WIMP***
Grafiskā lietotāja saskarne, kurā operētājsistēmu vadībai izmanto logus, ikonas un peli, kas būtiski atvieglo programmatūras lietošanu.

1740 **wire frame model**
**karkasmodelis**
**каркасная модель**
Trīsdimensiju objekta modelis, ko parasti izmanto datorgrafikā vai datorizētās projektēšanas sistēmās un kas tiek veidots no atsevišķām līnijām un lokiem.

1741 **wireless LAN**
**bezvadu lokālais tīkls**
**беспроводная локальная сеть**
Lokālais datoru tīkls, kura mezglu vai atsevišķu segmentu savienošanai izmanto nevis parastos savienotājkabeļus, bet gan radioviļņu vai infrasarkano staru tehnoloģiju.

1742 **wizard**
**vednis**
**оперативный консультант**
Interaktīvas palīdzības līdzeklis, ko sākotnēji firma *Microsoft* izveidoja lietojumiem operētājsistēmas *Microsoft Windows* vidē. Vednis palīdz lietotājam virzīties uz priekšu soli pa solim, uzstādot jautājumus, piedāvājot vajadzīgo informāciju un izskaidrojot iespējamos ceļus. Vedņi var būt dialoglodziņi, kuros attēlotas katrā solī iespējamās izvēles, vai arī tie var parādīties ekrānā kā kustīgi palīgi, kas laiku pa laikam piedāvā savu palīdzību.

1743 **word**
**vārds**
**слово**
Informācijas vienība, kas sastāv no rakstzīmēm, bitiem vai baitiem un ko apstrādā un uzglabā datora atmiņā kā vienotu veselumu.

1744 **word processing**
**tekstapstrāde**
**обработка текстов**
Datora izmantošana dokumentu veidošanā, rediģēšanā, formatēšanā, lasīšanā un drukāšanā. Tekstapstrāde ir viens no izplatītākajiem personālo datoru lietojumiem. Izstrādāts plašs tekstapstrādes programmu klāsts, kas nodrošina praktiski neierobežotas teksta rediģēšanas iespējas. Sk. arī *Word Perfect*, *Microsoft Word*.

1745 **word processing program**
**tekstapstrādes programma**
**программа обработки текстов**
Lietojumprogramma tekstapstrādei datorā. Lai atvieglotu teksta autora un redaktora darbu, jaunākajās tekstapstrādes programmās paredzētas plašas tekstu formatēšanas, pareizrakstības pār-

baudes, tekstu izguves un sagatavošanas, fontu un to lieluma izvēles iespējas. Sk. arī *text editor*.

1746 **word processor**
**tekstprocesors**
**текстовый процессор**
Sk. *word processing program*.

1747 **word wrap**
**aplaušana**
**перенос слова на новую строку**
Tekstapstrādes programmu spēja ietilpināt tekstu iepriekš noteiktā lappuses izmērā, automātiski pārtraucot rindiņas. Aplaušanas rezultātā izdarītos rindiņu pārtraukumus sauc par nestingro atgriezi.

1748 **WordPerfect**
**vārdu procesors *WordPerfect*, *WordPerfect***
**программа *WordPerfect***
Firmas *WordPerfect Corporation* 1980. g. izstrādāta tekstapstrādes programma, kas izmantojama operētājsistēmu *DOS*, *Microsoft Windows*, *OS/2*, *UNIX* vidē, kā arī *Macintosh* saimes datoros. Šajā programmā paredzētas plašas datorizdevniecības iespējas, ieskaitot importu, grafikas elementu izmēru maiņu, kā arī metodes *WYSIWYG* izmantošanu, kas dod iespēju lietotājam ekrānā iegūt tādu lappuses attēlu, kas atbilst tās drukātajam veidolam. Kopš 1994. g. firma *WordPerfect Corporation* ir kļuvusi par firmas *Novel* īpašumu.

1749 **worksheet**
**darblapa**
**рабочая таблица**
Izklājprogrammās — divdimensionāla rindiņu un kolonnu matrica, kurā var ievadīt nosaukumus, ciparus un formulas. Šo terminu bieži lieto termina "izklājlapa" vietā.

1750 **World Wide Web (WWW)**
**globālais tīmeklis,**
**tīmeklis *WWW*, *WWW***
**всемирная паутина,**
**глобальная гипертекстовая система *Internet***
Globāla hiperteksta sistēma, kas izmanto tīklu *Internet* kā informācijas transportēšanas mehānismu. Šo hiperteksta sistēmu veido informācija, kas sadalīta atsevišķās vienībās, kuras saistītas ar īpašām asociatīvām saitēm — hipersaitēm, tādējādi veidojot tīmekli. Novietojot kursoru uz kādas no hipersaitēm un noklikšķinot peli, hiperteksta sistēmas programmatūra izspīdina displeja ekrānā vajadzīgo informāciju. Šādu navigācijas procesu tīmeklī *WWW* parasti sauc par pārlūkošanu. Sk. arī *anchor*, *surfing*.

1751 **worm**
**tārps**
**вирусная программа самотиражирования**
Destruktīva pašreproducēšanās programma (datora vīrusa paveids), kas pakāpeniski aizpilda datora atmiņu, vairojoties tādās datora atmiņas vietās, kas netiek aktīvi izmantotas.

1752 **WORM (Write Once, Read Many)**
**daudzkārtlasāmā vienierakstes atmiņa, *WORM***
**atmiņa**
**однократно записываемая, многократно читаемая память**

Optiskā atmiņa ar lielu informācijas ierakstīšanas blīvumu un ietilpību, bet nemaināmu saturu. Informācijas ierakstīšanu un nolasīšanu *WORM* atmiņā veic ar lāzera stara palīdzību. Sk. arī *CD-ROM, magneto-optical disc.*

1753 **write protection**
**ierakstaizsardzība**
**защита от записи**
Informācijas dzēšanas vai ierakstīšanas aizliegums. Ierakstaizsardzību var izmantot gan lokanajam diskam, gan datnei lokanajā vai cietajā diskā.

1754 **write-protect notch**
**ierakstīšanas aizsargrobs**
**наклейка разрешения записи**
Neliels taisnstūrveida robs 5,25 collu lokanā diska malā, ko izmanto, lai to aizsargātu pret nesakcionētu vai nejaušu informācijas ierakstīšanu, kas var izraisīt vajadzīgo datu zudumu. Ierakstaizsardzību realizē, aizsedzot ierakstīšanas aizsargrobu ar īpašu uzlīmi. Sk. arī *write-protect tab.*

1755 **write-protect tab**
**ierakstīšanas aizsargbīdnis**
**прорезь разрешения записи**
Neliela atvere 3,5 collu disketes plastmasas apvalka stūrī, ko izmanto, lai aizsargātu disketi pret nejaušu informācijas ierakstīšanu, kas var radīt vajadzīgo datu zudumu. Lai veiktu ierakstaizsardzību, šo atveri aizklāj ar bīdni, kas speciāli ierīkots disketes plastmasas apvalkā. Sk. arī *write protect notch.*

1756 **WWW**
Sk. *World Wide Web.*

1757 **WYSIWYG**
**metode *WYSIWYG***
**метод *WYSIWYG***
Attēlu veidošanas metode displeja ekrānā, ko izmanto galvenokārt tekstapstrādē un datorizdevniecībā un kas dod iespēju lietotājam ekrānā iegūt tādu sagatavojamās lappuses attēlu, kas pēc sava izskata (fontiem, formāta, grafiku izkārtojuma utt.) atbilst tās drukātajam veidolam.


1758 **X-Window System**
**sistēma *X-Window***
**система *X-Window***
Saskarnes *API* instrukciju un rutīnu standartkopa, kas izstrādāta Masačūsetsas tehnoloģiskajā institūtā (ASV). Lai gan sistēma *X-Window* savā oriģinālajā variantā domāta darbstacijām, kas strādā ar operētājsistēmu *UNIX*, tā dod iespēju izveidot aparatūrneatkarīgu grafisko lietotāja saskarni.

1759 **X-Windows**
Sk. *X-Window System.*

1760 **Xerox**
**firma *Xerox***
**фирма *Xerox***
ASV firma, kas ir plaši pazīstama kā personālo datoru ārējo iekārtu ražotāja (peles, printeri ar iebūvētu procesoru u.c.). Firma 1976. g. izstrādāja lokālo tīklu *Ethernet*. Firmas *Xerox Palo Alto* pētniecības centrā (*Xerox PARC*) likti pamati grafiskās saskarnes (peles un logu) koncepcijai, kā arī izstrādāta viena no pirmajām objektorientētajām valodām *Smalltalk.*

1761 **Xerox PARC (Xerox Palo Alto Research Center)**
*Sk. Xerox*

1762 **XGA**
Sk. *eXtended Graphics Array.*

1763 **XMS**
Sk. *eXtended Memory Specification.*

1764 **XT**
Sk. *eXtended Technology.*

1765 **XT keyboard**
**tastatūra *XT***
**клавиатура *XT***
Personālā datora *PC/XT* tastatūra ar 83 taustiņiem. Tastatūra *XT* no tastatūras *AT* atšķiras galvenokārt ar to, ka tai ir 10 funkcionālie taustiņi, kas divās kolonnās izvietoti tastatūras kreisajā pusē.

## Y

1766 **yoke**
**nolieces spole**
**отклоняющая катушка**
Elektromagnētiska spole, kura vada elektronu stara kustību katodstaru lampās.

## Z

1767 **zoom box**
**mērogmaiņas lodziņš**
**окно масштабирования**
Neliels lodziņš aktīvā loga virsrakstjoslas labā augšējā stūrī. Noklikšķinot peli mērogmaiņas lodziņā, palielinās aktīvā loga izmēri un lietotājs var redzēt visu tā saturu. Pēc atkārtotas peles noklikšķināšanas logs atgūst savu iepriekšējo lielumu.

1768 **zooming**
**mērogmaiņa**
**изменение масштаба изображения**
Datorgrafikā — izvēlētā loga vai grafiskā attēla palielināšana visa ekrāna lielumā, kas zīmēšanas programmās atvieglo sīku detaļu koriģēšanu.

**Alfabētiskais rādītājs**

A/S "Poligrāfists", K. Valdemāra ielā 6, LV-1010.
Pasūt. nr. 183.