LATVIJAS UNIVERSITĀTES
RAKSTI

787. SĒJUMS

Datorzinātne un informācijas tehnoloģijas

SCIENTIFIC PAPERS
UNIVERSITY OF LATVIA

VOLUME 787

Computer Science and Information Technologies

SCIENTIFIC PAPERS
UNIVERSITY OF LATVIA

VOLUME 787

Computer Science and Information Technologies

University of Latvia

LATVIJAS UNIVERSITĀTES
RAKSTI
787. SĒJUMS

Datorzinātne un informācijas tehnoloģijas

Latvijas Universitāte

UDK 004(082)
Da 814

ISSN 1407-2157
ISBN 978-9984-45-569-3

Contents

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2012. Vol. 787
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 7–26 P.

Efficiency Measurements of Self-Testing

Edgars Diebelis

University of Latvia, Raina blvd. 19, Riga, Latvia, *edgars.diebelis@di.lv*

This paper is devoted to efficiency measurements of self-testing, which is one of smart technology components. The efficiency measurements were based on the statistics on incidents registered for eight years in a particular software development project: Currency and Securities Accounting System. Incident notification categories have been distributed into groups in the paper: bugs that would be identified, with appropriately located test points, already in the development environment, and bugs that would not be identified with the self-testing approach neither in the development, testing or production environments. The real measurements of the self-testing approach provided in the paper prove its efficiency and expediency.

Keywords. Testing, Smart technologies, Self-testing.

Introduction

Self-testing is one of the features of smart technologies [1]. The concept of smart technologies proposes to equip software with several built-in self-regulating mechanisms, for examples, external environment testing [2, 3], intelligent version updating [4], integration of the business model in the software [5] and others. The necessity of this feature is driven by the growing complexity of information systems. The concept of smart technologies is aiming at similar goals as the concept of autonomous systems developed by IBM in 2001 [6, 7, 8], but is different as for the problems to be solved and the solution methods. Autonomic Computing purpose [9] is to develop information systems that are able of self-management, in this way taking down the barriers between users and the more and more complex world of information technologies. IBM outlined four key features that characterise autonomic computing:

- Self-configuration. Automated configuration of components and systems follows high-level policies. Rest of system adjusts automatically and seamlessly.
- Self-optimization. Components and systems continually seek opportunities to improve their own performance and efficiency.
- Self-healing. System automatically detects, diagnoses, and repairs localized software and hardware problems.

- Self-protection. System automatically defends against malicious attacks or cascading failures. It uses early warning to anticipate and prevent system-wide failures.

With an aim similar to an autonomic computing, the smart technologies approach was offered in 2007 [1]. It provided a number of practically implementable improvements to the functionality of information systems that would make their maintenance and daily use simpler, taking us nearer to the main aim of autonomic computing. The main method offered for implementing smart technologies is not the development of autonomous universal components but the direct implementation of smart technologies in particular programs.

The first results of practical implementation of smart technologies are available. Intelligent version updating software is used in the Financial and budget report (FIBU) information system that manages budget planning and performance control in more than 400 government and local government organisations with more than 2000 users [4]. External environment testing [3] is used in FIBU, the Bank of Latvia and some commercial banks (VAS Hipotēku un zemes banka, AS SEB banka etc.) in Latvia. The third instance of the use of smart technologies is the integration of a business model and an application [5]. The implementation is based on the concept of Model Driven Software Development (MDSD) [10], and it is used in developing and maintaining several event-oriented systems.

Self-testing provides the software with a feature to test itself automatically prior to operation. The purpose of self-testing is analogical to turning on the computer: prior to using the system, it is tested automatically that the system does not contain errors that hinder the use of the system.

The article continues to deal with the analysis of the self-testing approach. Previous papers [11, 13, 14] have dealt with the ideology and implementation of the self-testing approach. As shown in [11, 12], self-testing contains two components:

- Test cases of system's critical functionality to check functions which are substantial in using the system;
- Built-in mechanism (software component) for automated software testing (regression testing).

In order to ensure development of high quality software, it is recommendable to perform testing in three phases in different environments: development, test, production [11]. Self-testing implementation is based on the concept of Test Points [13, 14]. A test point is a programming language command in the software text, prior to execution of which testing action commands are inserted. A test point ensures that particular actions and field values are saved when storing tests and that the software execution outcome is registered when tests are executed repeatedly. Key components of the self-testing software are: test control block for capturing and playback of tests, library of test actions, test file (XML file) [13, 14].

This paper analyses the efficiency of self-testing. Since it is practically impossible to test a particular system with various methods and compare them (e.g. manual

tests, tests using testing support tools with various features) with tests performed using the self-testing approach, another methodology for evaluating the efficiency has been used. The implementation of a comparatively large system was selected, and an expert analysed what bugs have been detected and in what stage of system development they could be detected, assuming that the self-testing functionality in the system would had been implemented from the very beginning of developing it. The analysis was based on the statistics of incident notifications registered during the system development stage, in which testing was done manually. The paper contains tables of statistics and graphs distributed by types of incident notifications, types of test points and years; this makes it possible to make an overall assessment of the efficiency of self-testing.

The paper is composed as follows: Chapter 1 briefly outlines the Currency and Securities Accounting System in order to give an insight on the system whose incident notifications were used for measuring the efficiency of the self-testing approach. Chapter 2 deals in more detail with the approach used for measuring the efficiency of self-testing. Chapter 3 provides a detailed overview of the measurement results from various aspects, and this makes it possible to draw conclusions on the pros and cons of self-testing.

1. Currency and Securities Accounting System (CSAS)

Since the paper's author has managed, for more than six years, the maintenance and improvement of the Currency and Securities Accounting System (CSAS), it was chosen as the test system for evaluating the self-testing approach. The CSAS, in various configuration versions, is being used by three Latvian banks:

- SEB banka (from 1996);
- AS Reģionālā investīciju banka (from 2007);
- VAS Latvijas Hipotēku un zemes banka (from 2008),

The CSAS is a system built in two-level architecture (client-server), and it consists of:

- client's applications (more than 200 forms) developed in: Centura SQL Windows, MS Visual Studio C#, MS Visual Studio VB.Net;
- database management system Oracle 10g (317 tables, 50 procedures, 52 triggers, 112 functions).

The CSAS consists of two connected modules:

- securities accounting module (SAM);
- currency transactions accounting module (CTAM).

1.1. Securities Accounting Module (SAM)

The software's purpose is to ensure the execution and analysis of transactions in securities. The SAM makes it possible for the bank to register customers that

hold securities accounts, securities and perform transactions in them in the name of both the bank and the customers. Apart from inputting and storing the data, the system also makes it possible to create analytical reports.

The SAM ensures the accounting of securities held by the bank and its customers and transactions in them. Key functions of the SAM are:

- Inputting transaction applications. The SAM ensures the accounting of the following transactions: selling/buying securities, security transfers, transactions in fund units, securities transfers between correspondent accounts, deregistration of securities, repo and reverse repo transactions, encumbrances on securities, trade in options;
- Control of transaction application statuses and processing of transactions in accordance with the determined scenario;
- Information exchange with external systems. The SAM ensures information exchange with the following external systems: the bank's internet banking facility, fund unit issuer's system, the Latvian Central Depository, the Riga Stock Exchange, the bank's central system, Bloomberg, SWIFT, the Asset Management System (AMS);
- Registration and processing of executed transactions;
- Calculation and collection of various fees (broker, holding etc);
- Revaluation of the bank's securities portfolio, amortisation and calculation of provisions;
- Control of partner limits;
- Comparing the bank's correspondent account balances with the account statements sent by the correspondent banks;
- Making and processing money payment orders related to securities. The SAM ensures the accounting of the following payments: payment of dividends (also for deregistered shares), tax collection/return, coupon payments, principal amount payments upon maturity;
- Making reports (inter alia to the Financial and Capital Market Commission, the Bank of Latvia, securities account statements to customers).

The SAM ensures swift recalculation of securities account balances based on transactions. The SAM is provided with an adjustable workplace configuration.

1.2. Currency Transactions Accounting Module (CTAM)

The purpose of the software is to account and analyse currency transactions (currency exchange, interbank deposits etc). The system accounts currency transactions and customers who perform currency transactions. Apart from data accounting, the system provides its users with analytical information on customers, currency transactions and currency exchange rates.

Key functions of the CTAM are:

- Inputting transactions. The CTAM ensures the accounting of the following transactions: interbank FX, FX Forward, SWAP (risk, risk-free), interbank depositing, (in/out/extension); customer FX, FX Forward, SWAP (risk, risk-free), interbank depositing, (in/out/extension), floating rate, interbank order, customer order, interest rate swaps, options, State Treasury transactions, collateral transactions, currency swap transactions;
- Control of transaction status and processing of transactions in accordance with the determined scenario (workflow);
- Information exchange with external systems. The CTAM ensures information exchange with the following external systems: bank's central system, Reuter, UBS TS trading platform, SWIFT;
- Maintaining currency positions (bank's total position, positions per portfolios);
- Setting and controlling limits;
- Importing transactions from Reuter and Internet trading platforms (TS, UBS TS etc);
- Importing currency exchange rates and interest rates from Reuters;
- Margin trading;
- Nostro account balances;
- Making reports.

The CTAM is provided with an adjustable workplace configuration.

2. Self-testing Efficiency Measurements Approach

All incident notifications (1,171 in total) in the CSAS in the period from July 2003 to 23 August 2011 retrieved from the Bugzilla [15] incident logging system were analysed. Since the incident logging system is integrated with a work time accounting system, in the efficiency measurements not only the quantity of incidents registered but also the time consumed to resolve incidents was used. Every incident notification was evaluated using the criteria provided in Table 1. As it can be seen in the table, not all incident notifications have been classified as bugs. Users register in the incident logging system also incidents that, after investigating the bug, in some cases get reclassified as user errors, improvements or consultations.

Table 1

Types of Incident Notifications

Type of Incident	Description
Unidentifiable bug	Incident notifications that described an actual bug in the system and that would not be identified by the self-testing approach if the system had a self-testing approach tool implemented.
Identifiable bug	Incident notifications that described an actual bug in the system and that would be identified by the self-testing approach if the system had a self-testing approach tool implemented.
Duplicate	Incident notifications that repeated an open incident notification
User error	Incident notifications about which, during resolving, it was established that the situation described had occurred due to the user's actions.
Improvement	Incident notifications that turned out to be system functionality improvements.
Consultation	Incident notifications that were resolved by way of consultations.

For measuring the efficiency of the self-testing approach, the most important types of incident notifications are Identifiable Bug and Unidentifiable Bug. Therefore, these types of incident notifications are looked at in more detail on the basis of the distribution of incidents by types provided in the tables below (Table 2 and Table 3).

Table 2

Unidentifiable Bug; Distribution of Notifications by Bug Types

Bug type	Description
External interface bug	Error in data exchange with the external system
Computer configuration bug	Incompliance of user computer's configuration with the requirements of the CSAS.
Data type bug	Inconsistent, incorrect data type used. Mismatch between form fields and data base table fields
User interface bug	Visual changes. For example, a field is inactive in the form or a logo is not displayed in the report.
Simultaneous users actions bug	Simultaneous actions by multiple users with one record in the system.
Requirement interpretation bug	Incomplete customer's requirements. Erroneous interpretation of requirements by the developer.
Specific event	Specific uses of the system resulting in a system error.

Table 3

Identifiable Bug; Distribution of Notifications by Test Point Types

Test point that would identify the bug	**Description**
File result test point	The test point provides the registration of the values to be read from and written to the file, inter alia creation of the file.
Input field test point	This test point registers an event of filling in a field.
Application event test point	This test point would register any events performed in the application, e.g. clicking on the Save button;
Comparable value test point	This test point registers the values calculated in the system. The test point can be used when the application contains a field whose value is calculated considering the values of other fields, the values of which are not saved in the database.
System message test point	This test point is required to be able to simulate the message box action, not actually calling the messages.
SQL query result test point	This test point registers specific values that can be selected with an SQL query and that are compared in the test execution mode with the values selected in test storing and registered in the test file.

3. Results of Measurements

3.1. Distribution of Notifications by Incident Types and Time Consumed

Table 4 shows a distribution of all incident notifications by incident types and the time consumed to resolve them in hours. The table contains the following columns:

- Notification type – incident notification type (Table 1);
- Quantity – quantity of incident notifications
- % of total – percentage of a particular notification type in the total quantity of incident notifications;
- Hours – total time consumed to resolve one type of incident notifications;
- % of total – percentage of the time spent to resolve the particular notification type of the total time spent for resolving all incident notifications.

Table 4

Distribution of All Notifications by Incident Types and the Time Consumed to Resolve Them

Application type	Quantity	% of total	Hours	% of total
Duplicate	68	5.81	23.16	0.47
User error	43	3.67	67.46	1.37
Unidentifiable bug	178	15.2	1,011.96	20.52
Identifiable bug	736	62.85	3,293.74	66.79
Improvement	102	8.71	241.36	4.89
Consultation	44	3.76	293.92	5.96
Total:	**1,171**	**100**	**4,931.6**	**100**

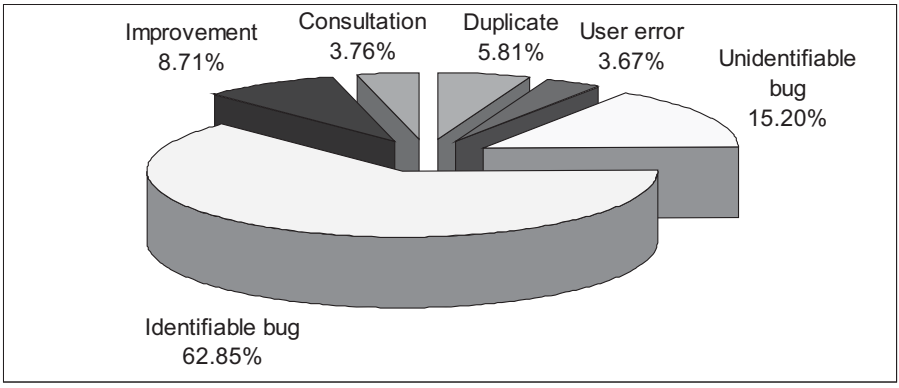


Figure 1. Distribution of All Notifications by Incident Types

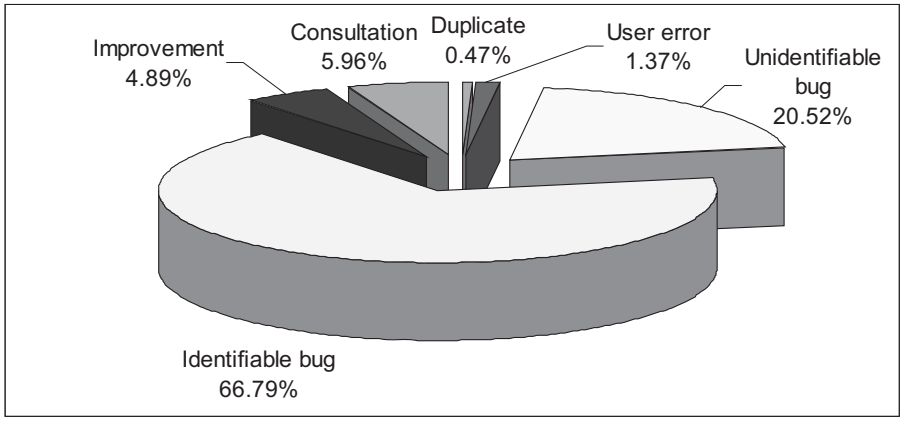


Figure 2. Distribution of All Notifications by the Time Consumed to Resolve per Incident Type

Considering the information obtained, the author concludes that:

- Of the total quantity of incident notifications (1,171), if the self-testing approach had been implemented in the CSAS, 62.85% (736 notifications, 3,293.74 hours consumed for resolving) of all the incident notifications registered in nearly nine years would had been identified by it already in the development stage. It means that developers would had been able to identify and repair the bugs already in the development stage, which would significantly improve the system's quality and increase the customer's trust about the system's quality. Measurements similar to [16] show the advantages of systematic testing compared to manual.
- As mentioned in many sources [17], the sooner a bug is identified, the lower the costs of repairing it. The differences mentioned reach to ten and even hundred times. The time spent for repairing bugs as provided in Table 4 would be definitely lower if the bugs had been identified already in the development stage. Assuming that the identification of bugs in the development stage would allow saving 50% of the time consumed for repairing all the bugs identified by the customer, about 1,650 hrs, or about 206 working days, could have been saved in the period of nine years.
- Of the total quantity of incident notifications, the self-testing approach in the CSAS would not be able to identify 15.2% of the bugs (178 notifications, 1,011.74 hours consumed for repairing) of the total number of incident notifications.
- Of the total quantity of incident notifications, 78.05% (914 notifications, 4,305.7 hours consumed for repairing) were actual bugs that were repaired, other 11.95% (257 notifications, time consumed for repairing 625.9 hours) of incident notifications were user errors, improvements, consultations and bug duplicates that cannot not be identified with testing.
- The time consumed for repairing bugs in percentage (87.31%) of the total time consumed for incident notifications is higher than the percentage (78.05%) of bugs in the total quantity of incident notifications. This means that more time has been spent to repair bugs proportionally to other incident notifications (improvements, user errors, consultations to users and bug duplicates).

3.2. Distribution of Notifications by Bug Types

As mentioned in the previous Chapter, there are two types of incident notifications that are classified as bugs: Unidentifiable Bugs and Identifiable Bugs, and they are analysed in the next table (Table 5). The table columns' descriptions are the same as in the previous Chapter.

Table 5

Distribution of Bugs by Bug Types

Bug Type	Quantity	% of total	Hours	% of total
Unidentifiable bug	178	19.47	1011.96	23.5
Identifiable bug	736	80.53	3293.74	76.5
Total:	**914**	**100**	**4305.7**	**100**

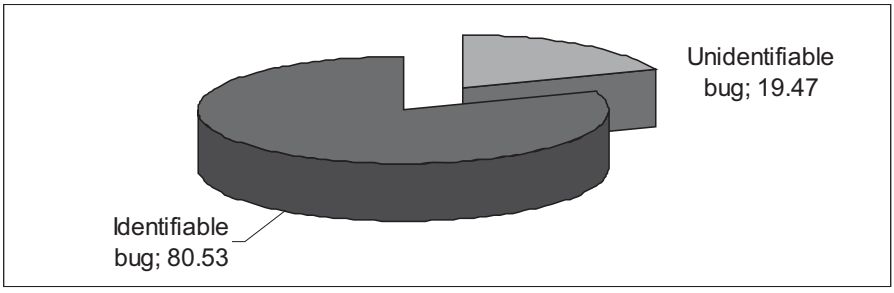


Figure 3. Distribution of Bugs by Bug Types

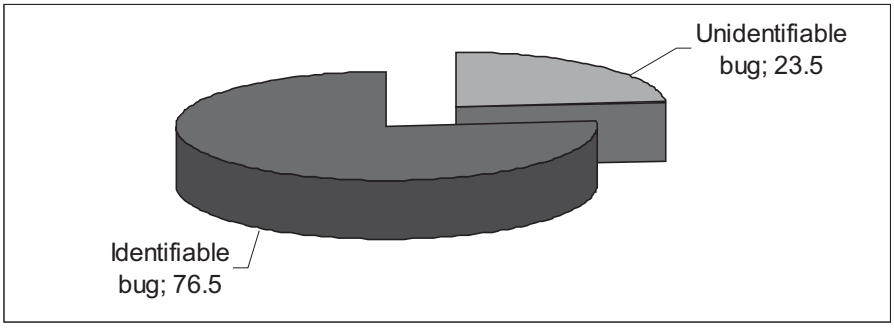


Figure 4. Distribution of Bugs by Bug Types per Time Consumed

Considering the information obtained, the author concludes that:

- The self-testing approach would identify 80% of all the bugs registered in the CSAS.
- The time consumed for repairing the bugs identified by the self-testing approach in percent (76.5%) of the total time consumed for resolving is lower than the percentage (80.53%) of these bugs in the total quantity of bugs. This means that less time would be spent to repair the bugs identified with the self-testing approach proportionally to the bugs that would not be identified with self-testing.

3.3. Distribution of Notifications by Years

The next table (Table 6) shows the distribution of incident notifications by years. The table contains the following columns:

- Notification type
  - quantity of incident notifications by type (Table 1) per years;
  - % of total – percentage of a particular notification type in the total quantity of incident notifications per year;
- 2003-2011 – years analysed;

Table 6

Distribution of Notifications by Years

Notification type	**2003**	**2004**	**2005**	**2006**	**2007**	**2008**	**2009**	**2010**	**2011**
Duplicate	1	1	13	22	7	10	7	5	2
% of total	3.23	1.3	13.13	10.19	4.9	5.29	3.45	3.85	2.41
User error	2	6	3	10	1	7	5	3	6
% of total	6.45	7.79	3.03	4.63	0.7	3.7	2.46	2.31	7.23
Unidentifiable bug	2	7	11	19	19	37	38	23	22
% of total	6.45	9.09	11.11	8.8	13.29	19.58	18.72	17.69	26.51
Identifiable bug	17	51	59	138	98	110	141	79	43
% of total	54.84	66.23	59.6	63.89	68.53	58.2	69.46	60.77	51.81
Improvement	9	12	13	25	11	11	6	11	4
% of total	29.03	15.58	13.13	11.57	7.69	5.82	2.96	8.46	4.82
Consultation	0	0	0	2	7	14	6	9	6
% of total	0	0	0	0.93	4.9	7.41	2.96	6.92	7.23
Total:	**31**	**77**	**99**	**216**	**143**	**189**	**203**	**130**	**83**

From the table it can be seen that:

- In some years, there are peaks and falls in the number of some incident notification types; also, consultation-type incident notifications have been registered as from 2005, but their proportional distribution by years match approximately the total proportional distribution of notifications.
- In any of the year's most of the bugs notified could be identified with the self-testing approach.

3.4. Distribution of Bugs by Years

The next table (Table 7) shows the distribution of bugs separately. The table contains the following columns:

- Year – years (2003-2011) analysed;

- Udentifiable bug – number of udentifiable bugs by years;
- % of total – quantity of udentifiable bugs as a percentage of the total number of udentifiable bugs;
- Identifiable bug – number of identifiable bugs by years;
- % of total – quantity of identifiable bugs as a percentage of the total number of identifiable bugs;
- Total – totals of bug quantities.

Table 7

Distribution of Bugs by Years

Year	Unidentifiable bug	% of total	Identifiable bug	% of total	Total
2003	2	1.12	17	2.31	**19**
2004	7	3.93	51	6.93	**58**
2005	11	6.18	59	8.02	**70**
2006	19	10.67	138	18.75	**157**
2007	19	10.67	98	13.32	**117**
2008	37	20.79	110	14.95	**147**
2009	38	21.35	141	19.16	**179**
2010	23	12.92	79	10.73	**102**
2011	22	12.36	43	5.84	**65**
Total	**178**	**100**	**736**	**100**	**914**

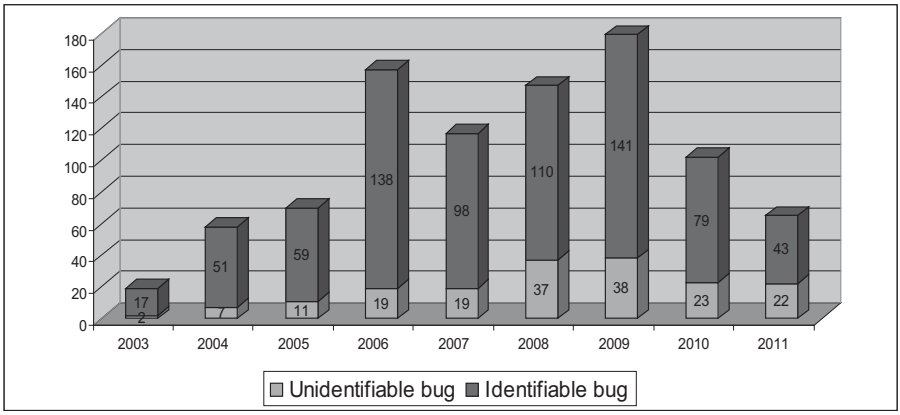


Figure 5. Distribution of Bugs by Years

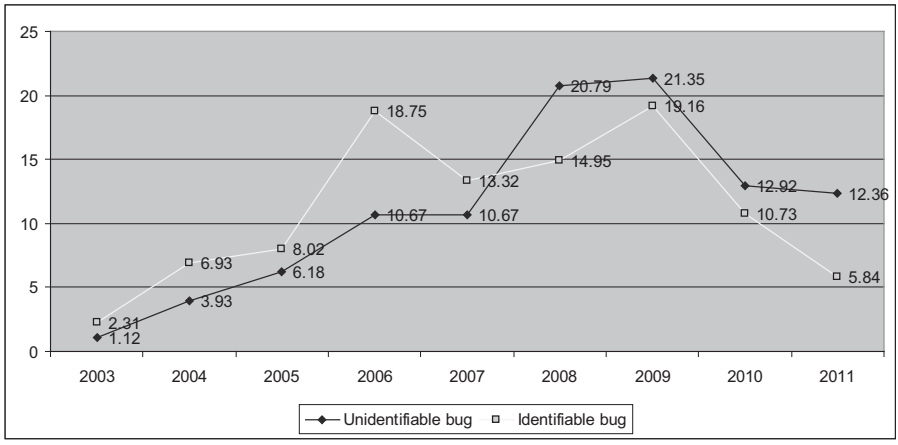


Figure 6. Distribution of Bugs in Percent by Years

Considering the information obtained, the author concludes that:

- The self-testing approach would identify most of the bugs registered in the CSAS.
- Changes, by years, in the percentages of the bugs identified and not identified with the self-testing approach are not significant.
- In the first five years, the weighting of the bugs that could be identified with the self-testing approach is higher, but later the weighting of the bugs that could not be identified with the self-testing approach becomes higher. This shows that the "simpler" bugs are discovered sooner than the "non-standard" ones.

3.5. Ratio of the Bug Volume to the Improvement Expenses Distributed by Years

The next table (Table 9) shows the ratio of the quantity of bugs to the volume of improvement expenses by years. The table contains the following columns:

- Year – years analysed;
- Quantity of bugs – quantity of bugs registered in the CSAS by years;
- % of total – distribution of the quantity of bugs in percent by years;
- Improvement expenses in % of total – percentage of the amount spent for improvements by years;

Table 8

Ratio of the Bug Quantity to the Improvement Expenses Distributed by Years

Year	Quantity of bugs	% to total	Improvement expenses in % of total
2003	31	2.65	2.86
2004	77	6.58	12.38
2005	99	8.45	15.25
2006	216	18.45	15.13
2007	143	12.21	11.04
2008	189	16.14	21.62
2009	203	17.34	9.77
2010	130	11.1	6.24
2011	83	7.09	5.71
Total:	**1,171**	**100**	**100**

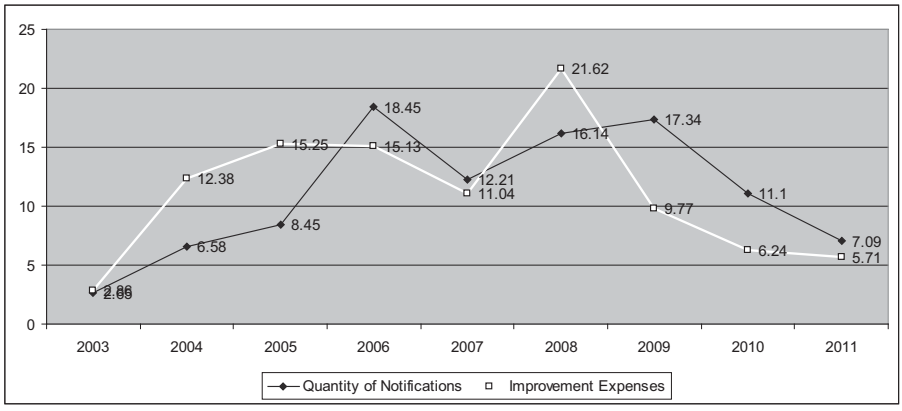


Figure 7. Ratio of the Quantity of Notifications to the Improvement Expenses

The information looked at in this Sub-chapter does not directly reflect the efficiency of the self-testing approach, but it is interesting to compare changes in the quantity of notifications (%) and in the improvement expenses (%) during the nine years. Considering the information obtained, it can be concluded that, as the improvement volumes grow, also the bug volumes grow. The conclusion is a rather logical one, but in this case it is based on an actual example.

3.6. Distribution of the Bugs Unidentifiable by the Self-Testing Approach by Types

The next table (Table 9) shows the distribution of bugs unidentifiable by the self-testing approach by bug types. The table contains the following columns:

- Bug type – one of seven bug types;
- Quantity – quantity of bugs of the type;
- % of total – percentage of the bug type in the total quantity of bugs.

Table 9

Distribution of the Bugs Unidentifiable by the Self-Testing Approach by Types

Bug type	**Quantity**	**% of Total**
External interface bug	5	2.81
Computer configuration bug	12	6.74
Data type bug	7	3.93
User interface bug	25	14.04
Simultaneous actions by users	5	2.81
Requirement interpretation bug	41	23.03
Specific event	83	46.63
Total:	**178**	**100**

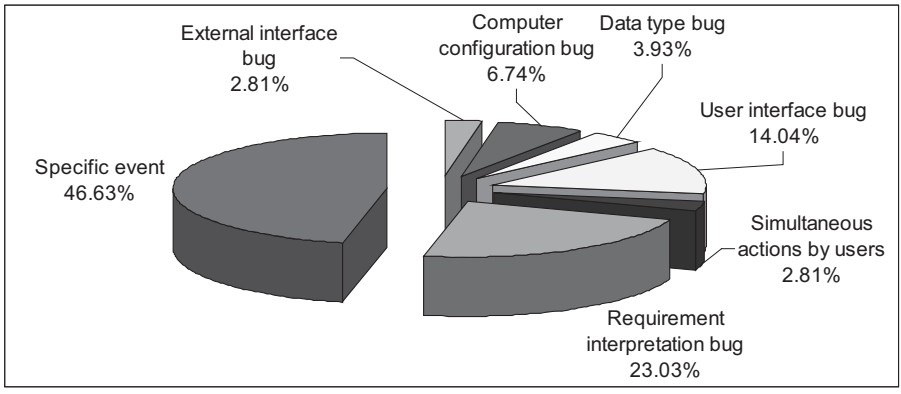


Figure 8. Distribution of the Bugs Unidentifiable by the Self-Testing Approach by Types

Conclusions:

- Most (nearly 50%) of the bugs that the self-testing approach would not be able to identify are specific cases that had not been considered when developing the system. For example, the following scenario from a bug description: "I enter the login, password, press Enter, press Enter once again, then I press Start Work, in the tree I select any view that has items under it. And I get an error!". As it can be seen, it is a specific case that the

developer had not considered. To test critical functionality, a test example that plans that Enter is pressed once would be made, and this test example would not result in a bug. Furthermore, a scenario that plans that Enter is pressed twice would not be created since it is a specific case not performed by users in their daily work. The self-testing approach is not able to identify bugs that occur due to various external devices. For example, when a transaction confirmation is printed, the self-testing approach would not be able to detect that one excessive empty page will be printed with it.

- One fifth of the total quantity of the bugs that the self-testing approach would be unable to identify are requirement interpretation bugs. Bugs of this type occur when system additions/improvements are developed and the result does not comply with the customer's requirements because the developer had interpreted the customer's requirements differently. To the author's mind, the quantity of hours (41) during the nine-year period is small and is permissible.
- The self-testing approach is unable to identify visual changes in user interfaces, data formats, field accessibility and similar bugs.
- A part of the registered bugs are related to incompliance of the user computer's configuration with the system requirements. The self-testing approach would be able to identify bugs of this type only on the user computer, not on the testing computer that has been configured in compliance with the system requirements.
- A part of the bugs that the self-testing approach would be unable to identify are data type bugs that include:
    - checking that the window field length and data base table field length match;
    - exceeding the maximum value of the variable data type.
- The self-testing approach is unable to identify bugs that result from the data of external interfaces with other systems. The self-testing approach is able to store and execute test examples that contain data from external interfaces, but it is unable to create test examples that are not compliant with the requirements of the external system (e.g. a string of characters instead of digits is given by the internal interface). Of course, it is possible to implement a control in the system itself that checks that the data received from the external interface are correct.
- The self-testing approach is unable to identify bugs that result from transaction mechanisms incorrectly implemented in the system, e.g. if several users can simultaneously modify one and the same data base record.

3.7. Distribution of the Bugs Identifiable by the Self-Testing Approach by Test Point Types

The next table (Table 10) shows the distribution of bugs identifiable by the self-testing approach by bug types and time consumed to resolve them. The table contains the following columns:

- Test point – test point that would identify the bug;
- Quantity – quantity of bugs that the test point would identify;
- % of total – percentage of the bugs that would be repaired by the test point in the total quantity of bugs that would be repaired by all test points;
- Hours – hours spent to resolve the bugs that the test points could identify;
- % of total – percentage of hours in the total number of hours consumed to repair the bugs that could be identified by test points.

Table 10

Distribution of the Bugs Identifiable by the Self-Testing Approach by Test Point Types

Test point	Quantity	% of total	Hours	% of total
File result test point	59	8.02	150.03	4.56
Entry field test point	146	19.84	827.14	25.11
Application event test point	105	14.27	364.24	11.06
Comparable value test point	28	3.8	93.53	2.84
System message test point	11	1.49	58.84	1.79
SQL query result test point	387	52.58	1,799.96	54.65
Total:	**736**	**100**	**3,293.74**	**100**

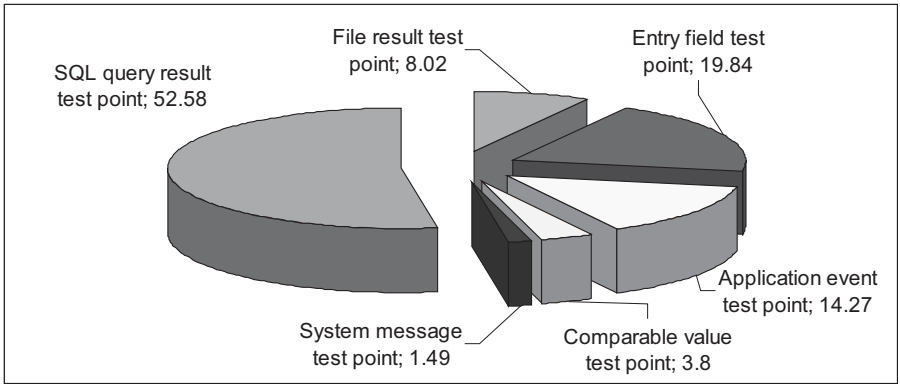


Figure 9. Distribution of the Bugs Identifiable by the Self-Testing Approach by Test Point Types

Conclusions:

- More than a half of all the registered bugs could be identified with the SQL query result test point. At the test point, data are selected from the data base and compared with the benchmark values. The explanation is that the key purpose of the CSAS is data storing and making reports using the stored data.
- One fifth of the bugs that could be identified with the self-testing approach would be identified by the input field test point. The test point compares the field value with the benchmark value.

4. Conclusions

In order to present advantages of self-testing, the self-testing features are integrated in the CSAS, a large and complex financial system. Although efforts are ongoing, the following conclusions can be drawn from the CSAS experience:

- Using the self-testing approach, developers would have been able to identify and repair 80% of the bugs already in the development stage; accordingly, 63% of all the received incident notifications would have never occurred. This would significantly improve the system's quality and increase the customer's trust about the system's quality.
- A general truth is: the faster a bug is identified, the lower the costs of repairing it. The self-testing approach makes it possible to identify many bugs already in the development stage, and consequently the costs of repairing the bugs could be reduced, possibly, by two times.
- Most of the bugs that the self-testing approach would be unable to identify are specific cases of system use.
- The SQL query result test point has a significant role in the identification of bugs; in the system analysed herein, it would had identified more than a half of the bugs notified.

From the analysis of the statistics, it can be clearly concluded that the implementation of self-testing would make it possible to save time and improve the system quality significantly. Also, the analysis has shown that the self-testing approach is not able to identify all system errors. On the basis of the analysis provided herein, further work in evolving the self-testing approach will be aimed at reducing the scope of the types of bugs that the current self-testing approach is unable to identify.

IEGULDĪJUMS TAVĀ NĀKOTNĒ

This work has been supported by the European Social Fund within the project «Support for Doctoral Studies at University of Latvia».

References

1. Bičevska, Z., Bičevskis, J.: Smart Technologies in Software Life Cycle. In: Münch, J., Abrahamsson, P. (eds.) Product-Focused Software Process Improvement. 8th International Conference, PROFES 2007, Riga, Latvia, July 2-4, 2007, LNCS, vol. 4589, pp. 262-272. Springer-Verlag, Berlin Heidelberg (2007).
2. Rauhvargers, K., Bicevskis, J.: Environment Testing Enabled Software - a Step Towards Execution Context Awareness. In: Hele-Mai Haav, Ahto Kalja (eds.) Databases and Information Systems, Selected Papers from the 8th International Baltic Conference, IOS Press vol. 187, pp. 169-179 (2009).
3. Rauhvargers, K.: On the Implementation of a Meta-data Driven Self Testing Model. In: Hruška, T., Madeyski, L., Ochodek, M. (eds.) Software Engineering Techniques in Progress, Brno, Czech Republic (2008).
4. Bičevska, Z., Bičevskis, J.: Applying of smart technologies in software development: Automated version updating. In: Scientific Papers University of Latvia, Computer Science and Information Technologies, vol .733, ISSN 1407-2157, pp. 24-37 (2008).
5. Ceriņa-Bērziņa J.,Bičevskis J., Karnītis Ç.: Information systems development based on visual Domain Specific Language BiLingva. In: Preprint of the Proceedings of the 4th IFIP TC 2 Central and East Europe Conference on Software Engineering Techniques, CEE-SET 2009, Krakow, Poland, Oktober 12-14, 2009, pp. 128-137.
6. Ganek, A. G., Corbi, T. A.: The dawning of the autonomic computing era. In: IBM Systems Journal, vol. 42, no. 1, pp. 5-18 (2003).
7. Sterritt, R., Bustard, D.: Towards an autonomic computing environment. In: 14th International Workshop on Database and Expert Systems Applications (DEXA 2003), 2003. Proceedings, pp. 694 - 698 (2003).
8. Lightstone, S.: Foundations of Autonomic Computing Development. In: Proceedings of the Fourth IEEE international Workshop on Engineering of Autonomic and Autonomous Systems, pp. 163-171 (2007).
9. Kephart, J., O., Chess, D., M. The Vision of Autonomic Computing. In: Computer Magazine, vol. 36, pp.41-50 (2003).
10. Barzdins, J., Zarins, A., Cerans, K., Grasmanis, M., Kalnins, A., Rencis, E., Lace, L., Liepins, R., Sprogis, A., Zarins, A.: Domain Specific languages for Business Process Managment: a Case Study Proceedings of DSM'09 Workshop of OOPSLA 2009, Orlando, USA.
11. Diebelis, E., Takeris, V., Bičevskis, J.: Self-testing - new approach to software quality assurance. In: Proceedings of the 13th East-European Conference on Advances in Databases and Information Systems (ADBIS 2009), pp. 62-77. Riga, Latvia, September 7-10, 2009.
12. Bičevska, Z., Bičevskis, J.: Applying Self-Testing: Advantages and Limitations. In: Hele-Mai Haav, Ahto Kalja (eds.) Databases and Information Systems, Selected Papers from the 8th International Baltic Conference, IOS Press vol. 187, pp. 192-202 (2009).

13. Diebelis, E., Bičevskis, J.: An Implementation of Self-Testing. In: Proceedings of the 9th International Baltic Conference on Databases and Information Systems (Baltic DB&IS 2010), pp. 487-502. Riga, Latvia, July 5-7, 2010.
14. Diebelis, E., Bicevskis, J.: Test Points in Self-Testing. In: Marite Kirikova, Janis Barzdins (eds.) Databases and Information Systems VI, Selected Papers from the Ninth International Baltic Conference. IOS Press vol. 224, pp. 309-321 (2011).
15. Bugzilla [Online] [Quoted: 20.05.2012] http://www.bugzilla.org/
16. Bičevskis, J.: The Effictiveness of Testing Models. In: Proc. of 3d Intern. Baltic Workshop "Databases and Information Systems", Riga, 1998.
17. Pressman, R.S., Ph.D., Software Engineering, A Practitioner's Approach. 6th edition, 2004.

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2012. Vol. 787
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 27–49 P.

Self-Testing Approach and Testing Tools

Valdis Vizulis, Edgars Diebelis

Datorikas Institūts DIVI, A. Kalniņa str. 2-7, Rīga, Latvia, *edgars.diebelis@di.lv*

The paper continues to analyse the self-testing approach by comparing features of the self-testing tool developed with those of seven globally acknowledged testing tools. The analysis showed that the features offered by the self-testing tool are equal to those provided by other globally acknowledged testing support tools, and outperform them in instances like testing in the production environment, testing of databases and cooperation in testing with external systems. Furthermore, the self-testing approach makes it possible for users with minimal IT knowledge to perform testing.

Keywords. Testing, Smart technologies, Self-testing, Testing tools.

Introduction

Already since the middle of the 20th century, when the first programs for computers were written, their authors have been stumbling on errors. Finding errors in programs was rather seen as debugging, not testing. Only starting from 1980s, finding errors in a program in order to make sure that the program is of good quality became the main goal of testing. [1]

Since then, software requirements and their complexity accordingly have grown constantly. Along the way, various testing methods, strategies and approaches have been developed. If years ago testing was done mainly manually, in our days, as system volumes and complexity grow, various automated solutions that are able to perform the process as fast as possible and consuming as possibly little resources are sought after.

One of ways of saving both time and resources consumed, improving the quality of the system at the same time, is the system self-testing approach [2]. This approach is one of smart technologies, and it enables the system to verify itself that the software is working correctly. Smart technology is based on the idea of software that is able to "manage itself" by ensuring a control over internal and external factors of the software and reacting to them accordingly. The concept of smart technologies besides a number of significant features also includes external environment testing [3, 4], intelligent version updating [5], integration of the business model in the soft-

ware [6]. The concept of smart technologies is aiming at similar goals as the concept of autonomous systems developed by IBM in 2001 [7, 8, 9].

The key feature of self-testing is the possibility to integrate the testing support option in the system to be tested, in this way ensuring automated testing at any time and in any of the following environments: development, testing and production. To demonstrate the usefulness of the self-testing approach, a self-testing tool that can be compared with globally popular testing tools has been developed. Therefore, the main goal of this paper was, by studying and comparing the concepts, builds and features of various globally recognized testing tools, to evaluate the usefulness of the self-testing approach and tool, directions for further development and opportunities in the area of testing.

As shown in [10, 11], self-testing contains two components:

- Test cases of system's critical functionality to check functions which are substantial in using the system;
- Built-in mechanism (software component) for automated software testing (regression testing) that provides automated storing and playback of tests (executing of test cases and comparing the test results with the standard values).

The defining of critical functionality and preparing tests, as a rule, is a part of requirement analysis and testing process. The self-testing software is partly integrated in the testable system [12, 13], which has several operating modes; one of them is self-testing mode when an automated execution of test cases (process of testing) is available to the user. After testing, the user gets a testing report that includes the total number of tests executed, tests executed successfully, tests failed and a detailed failure description. The options provided by self-testing software are similar to the functionality of testing support tools. Unlike them, the self-testing software is part of the system to be developed. It means that there is no need to install additional testing tools for system testing at the system developers, customers or users.

The paper is composed as follows: Chapter 1 describes the principles used to select the testing tools to be compared with the self-testing approach. Also, the selected testing tools are described in brief in this Chapter. Chapter 2 provides a description of the criteria used to compare the self-testing approach and the testing tools and a comparison of them.

1. Testing Tools

1.1. Selecting the testing tools

Nowadays a wide range of testing tools is available, and they are intended for various testing levels on different systems. When developing one's own testing tool,

it is important to find out what testing tools are being offered by other developers and what are their advantages and disadvantages.

Considering that the self-testing tool employs, in a direct way, the principles of automated testing, various automated testing tools were selected for comparing. In selecting the tools to be compared, the opinion of the Automated Testing Institute (ATI), which is a leading authority in the field of testing, was used. Since May 2009, the ATI has been publishing its magazine Automate Software Testing [14], which is one of the most popular in the field of testing and has its own website too [15]. The website offers articles by experienced IT professionals on the automated testing approach, frameworks and tools; the website has a list of 716 automated testing tools available in the world and brief descriptions of them. Also, a closed forum is active; its users are registered only after approval (currently there are about 8,000 users registered).

The ATI organises an annual conference on automated testing, called Verify/ ATI [16], during which new approaches and tools are demonstrated and training on them is provided. Since 2009, the company has been nominating the leading automated testing tools in various categories awarding them with the ATI Automation Honors [17]. Winners of the award are selected by a committee that is composed of IT professionals, and they study the tools (information on tools is obtained from their official websites, documentation, various articles, blogs, forums etc) and narrow down the list of tools applied for the award to five finalists in each category and sub-category (categories in 2010) [18]:

- Best open source code automated unit testing tool; sub-categories: C++, Java, .NET.
- Best open source code automated functional testing tool; sub-categories: WEB, Java, .NET, Flash/Flex.
- Best open source code automated performance testing tool; sub-categories: WEB, WEB services/SOA.
- Best commercial automated functional testing tool; sub-categories: WEB, Java, .NET, Flash/Flex, WEB services/SOA.
- Best commercial automated performance testing tool; sub-categories: WEB/HTTPS, WEB services/SOA.

For the comparison of the self-testing approach and testing tools in this paper, tools that have won an award in one of the aforementioned nominations were used. In the following sub-chapters of this paper, some of the award winners that are most similar to the self testing approach have been described in brief.

1.2. TestComplete

TestComplete is an automated self-testing tool developed by SmartBear; it provides the testing of Windows and web applications and is one of the leading functional testing tools in the world. This is also proven by the fact that the tool has won the ATI Automation Honors award as the Best Commercial Automated Functional Testing Tool in 2010, and it is used in their projects by world's leading companies like Adobe, Corel, Falafel Software, ARUP Laboratories, QlikTech etc. [19]

Concept

The TestComplete tool uses a keyword-driven testing framework to perform functional tests; in addition, with it it is possible to also develop tests with scripts. Its operation concept is comparatively simple. As shown in Figure 1, the tool, through inter-process communication and various built-in auxiliary tools, records the actions performed in the tested system and after that also executes them.

Figure 1. TestComplete Concept

After each test, the tool creates a detailed report on the test execution, showing the results of every command execution and the screenshots obtained during the playback. In this way, TestComplete makes it possible to overview the errors found in the test.

1.3. FitNesse

FitNesse is an open source code automated acceptance testing tool that can be used to create tests in the Wiki environment through cooperation among testers, developers and customers [20]. Wiki is a webpage content management system that makes it possible to create new or edit existing web pages with a text editor or a simple markup language [21].

In 2010, this tool won the ATI Automation Honors award as the best open source code automated functional testing tool in the NET sub-category.

FitNesse is based on the black box testing principles and Agile manifestos:

- People and interaction over processes and tools;
- Operating software over comprehensive documentation;
- Cooperation with the customer over negotiating the contracts;
- Reacting to changes over following the plan.

The goal of this tool is to make acceptance testing automated and easy to create and read also for people without in-depth IT knowledge. Consequently, customers themselves can develop their own tests as the test creation principles are, to the extent possible, tailored to the business logics. The engagement of the customer in the testing process helps to define the system requirements more precisely and the developers can better understand what the system has to do.

Concept

The testing concept of FitNesse is based on four components:

1. A Wiki page in which the test is created as a Decision Table;
2. A testing system that interprets the Wiki page;
3. A test fixture called by the testing system;
4. The tested system run by the test fixture. [22]

In the test development process, only the Wiki page and the test fixture has to be created over. Everything else is provided by the FitNesse tool and the tested system.

Depending on the test system used, FitNesse provides test tables of various types. To demonstrate the principles of how FitNesse works, the simplest test table, Decision Table, is looked at.

If it is required to develop a test which tests the class that performs the exponentiation of a number, then the following decision table has to be created on the Wiki page:

```
|Exponentiation|
|base|exponent|result?|
|2|5|32|
|4|2|16|
|1.5|2|2.25|
```

The Wiki environment transforms the text into a more illustrative format (Table 1).

Table 1

FitNesse Decision Table

ExponentiationTest		
base	**exponent**	**result?**
2	5	32
4	2	16
1.5	2	2.2

When executing the test, FitNesse delivers the table to the specified test system, which interprets it and calls the following test fixture created by the system developer (in this example, the test fixture is written in C#):

```
public class ExponentiationTest
{
   private double _base;
   private double _exponent;

   public void setBase(double base)
   {
      _base = base;
   }
```

```
    public void setExponent(double exponent)
    {
      _exponent = exponent;
    }

    public double result()
    {
      return TestedSystem.ExponentionClass.Exponent(_base, _exponent);
    }
  }
```

The interpreting is done between titles in the table and the test fixture code. The first row in the decision table contains the name of the test fixture class "ExponentiationTest". The second row specifies the test fixture class methods that are called in the same succession as the table columns are defined. For the methods followed by a question mark also the returned value is read and then compared with the expected value. Other methods set up the input data for the test fixture class. When the test is finished, the returned values are compared with the expected values, and the results are shown in the following format:

Table 2

FitNesse Decision Table after Test

ExponentiationTest		
base	**exponent**	**Result?**
2	5	32
4	2	16
1.5	2	2.2 expected 2.25 actual

Using this concept, system testing is generalised to business logics, and therefore the customers can participate in the process as they only have to additionally master the principles for creating Wiki pages and test tables, not a programming language. Furthermore, a Wiki page can be created also as the acceptance testing documentation since the tables demonstrate the criteria that must be fulfilled for the system developed to comply with the needs of the customer.

1.4. Ranorex

Ranorex is a typical graphic user interface testing tool that can be used by both testers and developers to swiftly and easy create new and manage existing functional tests. This tool has been appraised by the Automated Testing Institute: in 2010, this tool won its award as the best commercial automated functional testing tool in the NET and Flash/Flex sub-categories. In the entire category, it won the 2nd place after the aforementioned test tool TestComplete.

Customers that use this tool are globally known companies like Bosch, General Electrics, FujitsuSiemens, Yahoo, RealVNC etc.

Concept

Like all popular modern graphic user interface testing tools, also Ranorex's concept is based on keyword-driven testing, recording the object, action and identifier.

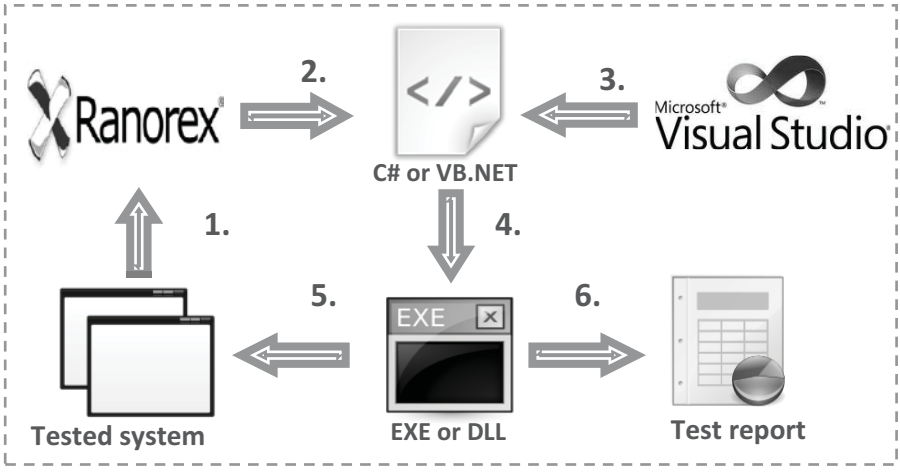


Figure 2. Ranorex Concept

1. Ranorex records the actions performed in the tested system.
2. Ranorex transforms the recorded actions into C# or VB.NET code.
3. To make the testing more convenient also for developers, the created C# or VB.NET code can be edited also in the Visual Studio development environment.
4. An executable file or library is compiled from the code.
5. The compiled testing "program" executes the recorded actions on the tested system.
6. When the test is finished, a test report is generated in which the test status is shown: successful or failed. For each test, detailed information can be viewed (Figure 3).

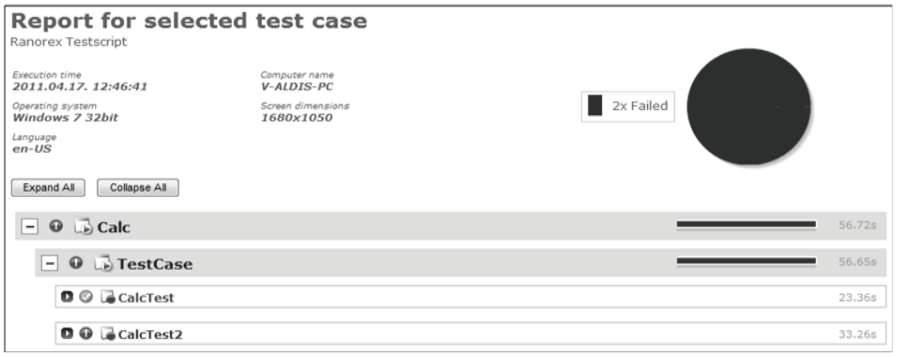


Figure 3. Ranorex Test Case Report

1.5. T-Plan Robot

T-Plan Robot is a flexible and universal graphic user interface testing tool that is built on image-based testing principles. This open source code tool does system testing from the user's perspective, i.e. visual. Since the tool is able to test systems that cannot be tested with tools that are based on the object-oriented approach, in 2010 this tool won the ATI Automation Honors award as the best open source code automated functional testing tool in the Java sub-category. Among the company's customers there are Xerox, Philips, Fujitsu-Siemens, Virgin Mobile and other.

Concept

Unlike many other typical functional testing tools, T-Plan Robot uses neither data- nor keyword-driven testing. Instead, it uses an image-based testing approach.

This approach lets the tool be independent from the technology which the tested system is built or installed on. This tool is able to test any system that is depicted on the operating system's desktop.

T-Plan Robot works with desktop images received from remote desktop technologies or other technologies that create images. For now, the tool only supports the testing of static images and the RFB protocol, which is better known with the name Virtual Network Computing. In future it is planned to add support for the Remote Desktop Protocol and local graphic card driver. [23]

In Figure 4, it can be seen how the testing runs using the client-server principle. The client and the server can cooperate through the network using the TCP/IP Protocol, or locally, using the desktop driver. T-Plan Robot works as a client that sends keyboard, mouse and clipboard events to the server. The tested system is located on the server, which sends to the client changes in the desktop image and the clipboard. T-Plan Robot is installed on the client's system and runs on a Java virtual machine, which makes the tool independent from the platform.

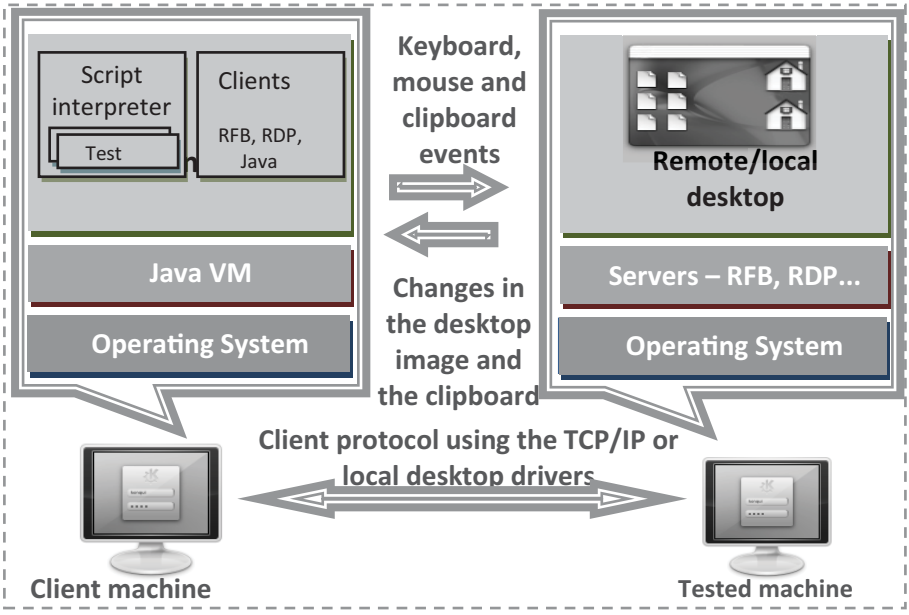


Figure 4. T-Plan Robot Concept [23]

Tests are recorded by connecting to the remote desktop and running the tested system. At this moment T-Plan Robot registers the sent input data and the received changes in the desktop image or the clipboard and creates a test script.

The test is played back by executing the test script that contains the input data to be sent to the tested system. The received changes in the desktop image and the clipboard are compared with those registered when recording the test. At this moment the tool's image comparison methods are used.

The **Client–Server architecture** can be provided in three various ways [23]:

1. One operating system with several desktops: only supported by Linux/Unix as it lets run several VNC servers simultaneously;
2. One computer wits several operating system instances: supported by all operating systems because, using virtualisation technologies (e.g. VirtualBox, VMware etc), the VNC server can be installed on the virtual computer;
3. Two separate computers: supported by all operating systems because when the computers are connected in a network, one runs as a client and the other as a server on which the tested system is installed.

1.6. Rational Functional Tester

The product offered by IBM, Rational Functional Tester (RFT), is an automated object-oriented approach automated functional testing tool that is one of the components in the range of lifecycle tools of the IBM Rational software.

This tool is one of the most popular testing tools, but it has not won any ATI Automation Honors awards. In 2009 and 2010, it was a finalist among the best commercial automated functional and performance testing tools. It shows that nowadays there appear more and more new and efficient automated testing tools to which also RFT is giving up its positions in the market of testing tools.

Concept

IBM RFT was created to ensure automated functional and regress testing for the testers and developers who require premium Java, NET and web applications testing.

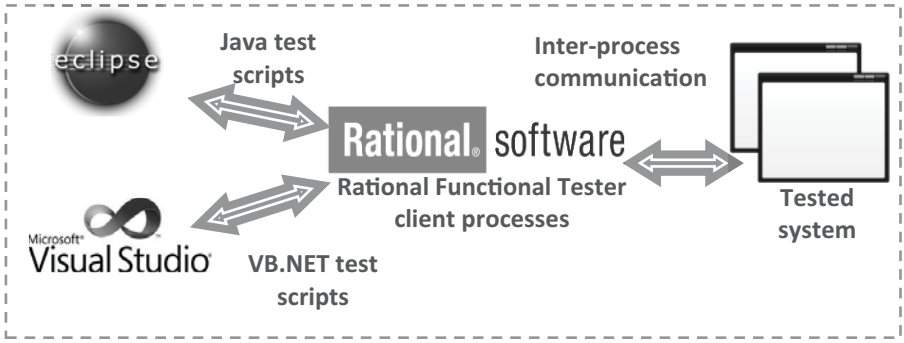


Figure 5. Rational Functional Tester Concept [24]

RFT does not have its own graphic user interface. Instead, RFT uses Eclipse, a development environment that Java users are well familiar with, or Visual Studio that is used by NET developers (Figure 5). In these development environments, RFT adds during installation an additional functionality that makes it possible to record, play back, edit and manage tests. [24]

When the test is recorded, all the actions that take place in the tested system are at once transformed by RFT into Java or VB.NET test scripts. During the recording process, the tester themselves has to create control points for RFT to register the expected system state (e.g. field value, object attribute, system screenshot etc) and later, when the test is played back, to compare that state with the current state.

When playing back the test, RFT executes the test scripts generated during the recording and compares the result of every executed action with the result of the recorded action. If test points are defined in the test script, RFT performs verification in relation to the expected system state defined in the test script. After the execution of each test script, RFT generates a HTML file (log) that demonstrates the test results and shows all the discrepancies for the expected system state.

1.7. HP Unified Functional Testing Software

HP Unified Functional Testing Software (HP UFTS), a tool offered by Hewlett Packard, is a premium quality automated functional and regress testing tool set that consists of two separate tools: HP Functional Testing Software (HP FTS) and HP Service Test Software (HP STS).

HP FTS is better known as HP QuickTest Professional (HP QTP) and formerly also as Mercury QuickTest Professional. HP FTS is based on HP QTP, but it has been supplemented with various extensions. It replaced a formerly popular tool, Mercury WinRunner, which was bought over by HP in 2006. Since most of the tool's functionalities overlapped (or were taken over to) with HP FTS, in 2008 it was decided to terminate the support to the tool and it was recommended to transfer any previously recorded tests to HP FTS. HP STS, in turn, is a tool developed by HP itself, and it ensures automated functional testing of services with the help of activities diagrams. [25]

By merging the tools, HP obtained one of the most popular functional and regress testing tool in the world; in 2009, this tool won the ATI Automation Honors award as the best commercial automated functional testing tool, and in 2010 it was among the four finalists in the same category.

Concept

HP UFTS is a typical keyword- and data-driven testing tool that both makes the creation and editing of tests easier and ensures wide coverage of tests for tested systems.

To perform complete functional testing of a system, it is not sufficient to test the graphic user interface as the system functionality is not limited to solely the visual functionality. Many functionalities are "hidden" under the graphic user interface on the level of components. It can be especially seen in systems that run according to the principle "client-server". These systems are called multi-level systems.

To ensure the testing of such systems, HP UFTS is based on the concept of multi-level testing (Figure 6). HP UFTS distributes multi-level testing in three levels [26]:

- graphic user interface testing;
- services and components testing;
- multi-level testing.

Graphic user interface testing is provided by HP FTS, which divides it into two levels: business processes testing and applications testing. Business processes testing can be done thanks to the test recording and playback functionality and keyword-driven testing that significantly simplify the creation and editing of tests and bring them closer to the business logics. Furthermore, to ensure quality testing of applications, HP FTS, with the integrated script creation and debugging

environment, offers full access to graphic user interface objects and their features. Consequently, HP FTS can be used to test both the process and the GUI level at the same time.

Testing of components and services is done by HP STS, which makes it possible to conveniently create functional tests for the invisible part of the tested system. Tests are created with the help of activities diagrams, and programming is only required for more complex tests; therefore, tests can be created also by users not having knowledge in programming.

Multi-level testing is done by HP UFTS, which combines HP FTS and HP STS in a single solution. HP UFTS can be used to test transactions that unites together the levels of the tested multi-level system. In this way, it is possible, in one test scenario, to test graphic user interfaces as well as services and components.

Test results are generated after the execution of every test in all three testing levels. Both HP FTS and HP STS show test results in two levels. The first level shows general test results and statistics, and the second level offers a detailed description of the results of executing every command and a comprehensive description of every error. HP UFTS gathers the test results in a single report, where all commands are distributed into either HP FTS or HP STS activities.

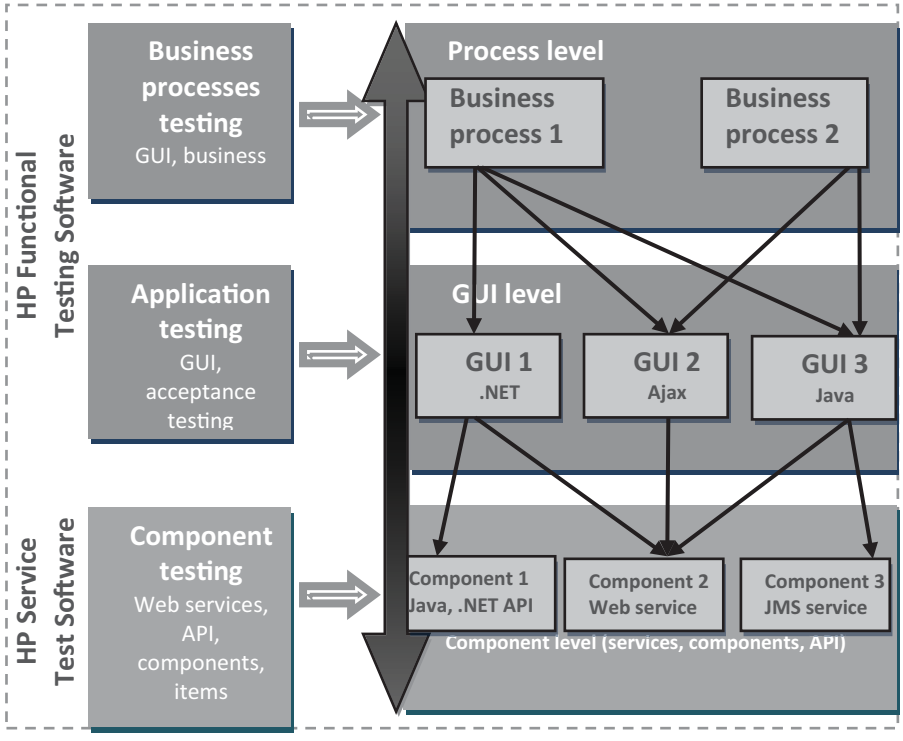


Figure 6. HP UFTS Multi-level Testing Concept [26]

As it can be seen in Figure 6, HP FTS enables the testing of both business processes and applications, making it possible to test the process and the GUI levels respectively in the tested system. Each business process is provided with one or more graphic user interfaces in various setups.

Graphic user interface is the visible part of the tested system that calls various components and services and receives from them the results to be showed to the users. As it can be understood, a majority of system functionalities are located on the system component level, and therefore HP STS offers the functional testing of various services, application interfaces, components and items.

From the aspect of testing concept, HP FTS does not offer a new approach, but in combination with HP STS this tool is able to offer different and diverse functional testing in three levels. This multi-level functional testing makes it possible to perform the testing prior to developing the graphic user interface, in this way allowing a faster development of the system and increasing the quality of components and services. Consequently, the overall quality of the graphic user interface is increased.

1.8. Selenium

Selenium is an open source code web application testing framework developed by OpenQA, and it consists of several testing tools. In the field of web applications testing, this framework has been a stable leader for more than five years, and last two years it has won the ATI Automation Honors award as the best open source code automated functional testing tool. A factor that contributes significantly to the advancement of this tool is that it is used in their testing projects by IT companies like Google, Mozilla, LinkedIn and others.

Concept

The Selenium framework consists of three different tools [27]:

- **Selenium IDE** is a Selenium script development environment that makes it possible to record, play back, edit and debug tests.
- **Selenium Remote Control** is a basic module that can be used to record tests in various programming languages and run them on any browser.
- **Selenium Grid** controls several Selenium Remote Control instances to achieve that tests can be run simultaneously on various platforms.

For test creation, Selenium has developed its own programming language, Selenese, which makes it possible to write tests in different programming languages. To execute tests, a web server is used that works as an agent between the browser and web requests, in this way ensuring that the browser is independent.

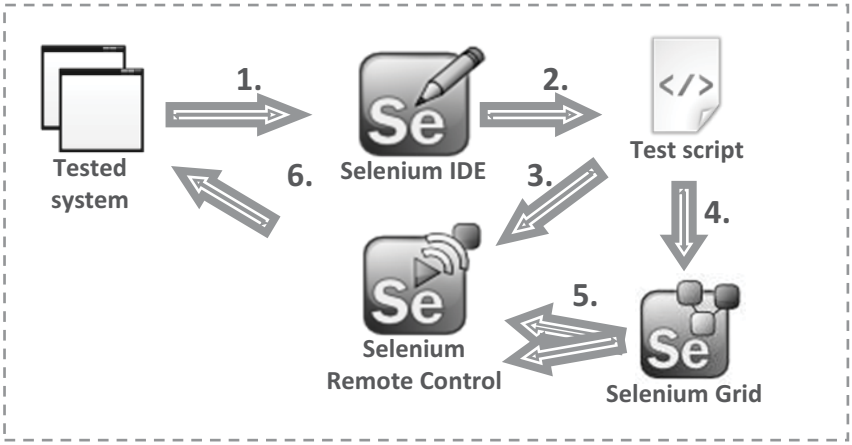


Figure 7. Selenium Concept [27]

1. Selenium IDE, using Mozilla Firefox browser, records the actions performed in the tested system.
2. All the recorded actions are recorded in the Selenese programming language, and if required it is possible to export them to C#, Java, Perl, PHP, Python or Ruby programming languages. After that, using a tool of any of these programming languages, existing script tests can be complemented or new test scripts can be created.
3. The commands recorded in the Selenium Remote Control test script are transformed into web requests.
4. Selenium Grid reads the test script and creates several Selenium Remote Control instances.
5. Selenium Grid simultaneously calls the created Selenium Remote Control instances feeding the read test script as a parameter.
6. Selenium Remote Control forwards to the tested system the web requests that comply with the commands defined in the test script.

It should be added that the concept of the Selenium framework does not require the succession shown in Figure 7. Depending on the knowledge and experience of the tester and the project's needs, the succession can be modified. To create the first tests, an inexperienced tester will certainly use the Selenium IDE development environment. In this case, the testing will be done using the complete cycle (scenarios in Figure 7 – 1, 2, 3 and 6 or 1, 2, 4, 5 and 6). To create more complex tests, an experienced tester will usually skip the first two steps as the creation of tests will be done in a development environment of a programming language supported by Selenium and the execution is provided for by Selenium Remote Control and Selenium Grid (scenarios in Figure 7 – 3, 6 or 4, 5, 6).

2. Comparison of the Self-Testing Tool with other Testing Tools

The aim of this paper is to evaluate what features are offered by the self-testing tool compared to other tools and to identify directions for further development. It is difficult to determine from voluminous descriptions of tools what advantages and disadvantages a tool has compared to other tools. Also, it is rather difficult to assess which tool is best suited for a certain testing project. For this reason, it is important to compare the tools using certain criteria, which are analysed hereinafter and on the basis of which the tools will be compared.

2.1. Criteria for comparison

The criteria for comparison were selected on the basis of the possibilities offered by those seven tools looked at herein. The following aspects were taken into account by the author in selecting the criteria for comparison:

1. key features of testing;
2. key features of automated testing tools;
3. features offered by the compared tools.

Table 3 below lists the criteria used to compare testing tools and the question arising from the criteria, and answers to the questions are provided in the tool comparison tables (Table 4 and Table 5). To compare the self-testing tool and other tools looked at in this paper, the criteria described in Table 3 were used.

Table 3

Criteria for Comparing Testing Tools

Comparison criteria	Question
Test method (TM) [28]	What test methods are supported?
Test automation approach (TAA) [29]	What test automation approaches are used?
Test automation framework (TAF) [30]	What test automation frameworks are used?
Testing level	What test levels are supported?
Functional testing	Is functional testing supported?
Non-functional testing	What non-functional testing aspects are supported?
Platform	What operating systems are supported?
Testable technology	What technologies (usually programming languages) are supported?
Test recording and playback	Is test recording and automated reiterated playback provided?

Desktop applications testing	Is desktop applications testing provided?
Web applications testing	Is web applications testing provided?
Services testing	Is services testing provided?
Database testing	Is it possible to test only the system database separately?
Testing in production environment	Is testing in the production environment provided?
System user can create tests	Can system users without in-depth IT knowledge create tests?
Simultaneous running of several tests	Can tests be run simultaneously?
Performing simultaneous actions	Will the test performance not be disturbed if during the test simultaneous actions are performed?
Identifying the tested object	Is it able to tell apart the object to be tested from other objects of the operating system?
Test result analysis	Is a test result analysis offered after the test?
Test editing	Is an editor for the created tests offered?
Screenshots	Are screenshots of the tested system acquired during the recording/playback of the test?
Control points	Are control points offered?
Object validation	If modifications take place in the tested system, is object validation provided?
Object browser	Is a browser/editor for objects of the tested system offered?
Test log	Is a test performance log created?
Test schedule planner	Is it possible to set the time for performing the test, e.g. at night?
Identification of the end of command execution	Is the tool able to determine when the execution of the previous command has ended (and a certain waiting time is not used for this purpose)?
Plug-ins and extensions	Is it possible to create own plug-ins and extensions to expand the tool's functionality?
Developer	What company or person has developed (owns) the testing tool?
Price	How much the tool costs?
Convenience of use (1-5)	On the scale from 1 (very inconvenient) to 5 (very convenient) – how convenient it is to create tests (author's subjective assessment)?
Tool programming language	In what programming languages it is possible to create tests?
Client	What companies use the tool in their testing projects?

2.2. Comparison results

To assess every comparison criteria determined, information obtained from the tool's official website and other trusted websites, specifications and help windows and from practical use of the test tools was used to make sure that they comply with the respective criteria.

To ensure maximum objectivity of the comparison results, the author tried, to the extent possible, avoid using his own subjective judgment and base the comparison on whether the tool offers a feature or not. To this end, the criteria in the comparison provided in Table 4 and table 5 were evaluated using the following three answers:

- Yes – it means that the tool supports the functionality referred to in the criteria;
- Partially – it means that the tool partially supports the functionality referred to in the criteria;
- No – it means that the tool does not support the functionality referred to in the criteria.

The paper contains two comparison tables: Table 4 provides a summary on the comparison criteria that the self-testing tool supports, and Table 5 provides a summary on the comparison criteria that the self-testing tool does not currently support.

Table 4

Comparison of Testing Tools (Features Provided by Self-Testing Tool)

Criteria		Self-testing tool	TestComplete	FitNesse	Ranorex	T-Plan Robot	Rational Functional Tester	Selenium	HP Unified Functional Testing Software
TM	Black-box	Yes	No	No	Yes	Yes	Yes	Yes	Yes
	White-box	Yes	No	No	No	No	No	No	No
	Grey-box	Yes	Yes	Yes	No	No	No	No	No
TAA	Object oriented	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes
TAF	Linear	Yes	No	No	No	Yes	No	No	No
Testing level	Unit	Yes	Yes	Yes	No	No	No	No	No
	Integration	Yes	Yes	Yes	Partially	Partially	Partially	Partially	Yes
	Regresion	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Acceptance	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Functional testing		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Platform	Win 2000	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
	Win Server 2003, 2008	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Win XP	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Win Vista	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Win 7	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Testable technology	.NET (C#, VB, C++)	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Test recording and playback		Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
Desktop applications testing		Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Database testing		Yes	Yes	Yes	No	No	No	No	Yes
Testing in production environment		Yes	No	No	No	No	No	No	No
System user can create tests		Yes	No	No	No	No	No	No	No
Simultaneous running of several tests		Yes	No	Yes	No	No	No	Yes	No
Performing simultaneous actions		Yes	No	Yes	No	No	No	No	Yes
Identifying the tested object		Yes	Yes	Yes	No	No	Yes	Yes	Yes
Test result analysis		Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Control points		Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
Identification of the end of command execution		Yes	Yes	Yes	No	No	Yes	Yes	Yes

Table 5

Comparison of Testing Tools (Features not Provided by Self-Testing Tool)

Criteria		Self-testing tool	TestComplete	FitNesse	Ranorex	T-Plan Robot	Rational Functional Tester	Selenium	HP Unified Functional Testing Software
TAA	Image-based	No	No	No	No	Yes	No	No	No
TAI	Data-driven	No	Yes	Yes	Yes	No	Yes	Partially	Yes
	Functional Decomposition	No	No	No	No	No	No	No	No
	Keyword-driven	No	Yes	Yes	Yes	No	Yes	Yes	Yes
	Model-based	No	No	No	No	No	No	No	No
Non-functional testing	Recovery	No	No	No	No	No	No	No	No
	Security	No	No	No	No	No	No	No	No
	Stress	No	Yes	No	No	No	No	No	No
	Load	No	Yes	No	No	No	No	No	No
	Performance	No	No	No	No	No	No	No	No
Platform	Pocket PC	No	Yes	Yes	No	Yes	No	No	No
	Win Mobile	No	Yes	No	No	Yes	No	No	Yes
	Unix	No	No	Yes	No	Yes	Partially	Yes	No
	OS X	No	No	Yes	No	Yes	No	Yes	No
	Other	No	No	No	No	Yes	No	Partially	No
Testable technology	Java	No	Yes	Yes	No	Yes	Yes	Yes	Yes
	Delphi	No	Yes	Yes	Yes	Yes	No	No	Yes
	ASP.NET	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Flash/Flex	No	Yes	No	Yes	Yes	Yes	Yes	Yes
	Silverlight	No	Yes	No	Yes	Yes	No	Yes	Yes
	HTML	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Ruby	No	No	Yes	No	Yes	No	Yes	No
	Python	No	No	Yes	No	Yes	No	No	No
	Perl	No	No	Yes	No	Yes	No	No	No
	Siebel	No	No	No	No	No	Yes	No	No
	SAP	No	No	No	No	No	Yes	No	Yes
	Other	No	No	No	No	Yes	No	Partially	No
Web applications testing		No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Services testing		No	No	No	No	No	No	No	Yes
Test editing		No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Screenshots		No	Yes	No	Yes	Yes	Yes	No	Yes
Object validation		No	Yes	No	Yes	No	Yes	Yes	Yes
Object browser		No	Yes	No	Yes	No	Yes	No	Yes
Test log		No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Test schedule planner		No	Yes	No	No	No	No	No	No
Plug-ins and extensions		No	Yes	Yes	No	Yes	Yes	Yes	Yes

The next table (Table 6) shows the additional criteria for comparing the tools that were not included in the previous tool comparison tables.

Table 6

Comparison of Testing Tools 2

Testing Tool	Developer	Price (EUR)	Con-veni-ence	Client	Tool 's programming language
Self-testing tool	Datorikas institūts DIVI	*	5	*	C#
TestComplete	SmartBear Software	~ 1 400	3	Adobe, Corel, Falafel Software, ARUP Laboratories, QlikTech u.c.	VBScript, Jscript, C++Script, C#Script, DelphiScript
FitNesse	Robert C. Martin	Free	4	*	Java
Ranorex	Ranorex	~ 1 190	3	Bosch, General Electrics, FujitsuSiemens, Yahoo, RealVNC u.c.	C++, Python, C#, VB.NET
T-Plan Robot	T-Plan	Free	3	Xerox, Philips, FujitsuSiemens, Virgin Mobile u.c.	Java
Rational Functional Tester	IBM	2 700 – 11 000	4	*	Java, VB.NET
Selenium	OpenQA	Free	3	Google, Mozilla, LinkedIn u.c.	C#, Java, Perl, PHP, Python, Ruby
HP Unified Functional Testing Software	Hewlett Packard	3 000 – 10 000	4	*	VBScript, C#

* – no/not available

3. Conclusions

To compare the seven testing tools, the author had to analyse not only their builds but also their concepts in order to assess objectively what are the pros and cons of the self-testing tool and what could be the directions for its further

development. From the comparison of the seven tools, the following conclusions can be drawn:

- The self-testing tool, TestComplete and FitNesse, thanks to the possibility to access the internal structure of the tested system, offers the grey-box testing (self-testing tool offers also white-box testing) method, which makes it possible to test the system more detailed. The other testing tools employ the black-box testing method;
- Among the seven tools looked at in this paper, only one uses an image-based approach as the test automatisation approach. All the others use the object-oriented approach. Consequently, it can be concluded that the object-oriented approach is the most popular for the automatisation of tests;
- In comparison to the other tools described in this paper, the self-testing tool provides for a wide range of testing levels, as TestComplete is the only one that, in addition to unit, integration, regress, functional and acceptance testing, offers also stress and load testing, while the others are limited to only three testing levels. It has to be noted that, considering the ability of self-testing to run parallel tests, it would be comparatively simple to implement the stress and load testing support also in the self-testing tool.
- Of the test automation tools dealt with in this paper, FitNesse is the only one that does not provide the test recording and playback functionality. Instead of it, a convenient creation of tests in table format is offered.
- Just a few testing tools offer such features included in the self-testing tool as database testing, simultaneous execution of tests and parallel actions during testing, whereas almost all testing tools offer the identification of the tested object, test result analysis and the adding of control points to the test;
- Only the self-testing tool offers the possibility to run testing in the production environment and the possibility to create tests for users without in-depth IT knowledge;
- The self-testing tools and Ranorex are the only one that do not offer the feature of adding existing or new plug-ins and extensions;
- The self-testing tool is not the only one that requires additional resources prior to creating tests, as also for FitNesse test fixtures must be developed for successful performance of the tests.

A number of the criteria listed in table 5 and currently not supported by the self-testing tool can be implemented through comparatively minor improvements to the tool. For example, to achieve that the self-testing tool supports performance testing, just a new test point that would control the performance of action execution needs to be implemented.

Self-testing is a new and original approach that does not lag behind other tools, and in some areas it is undoubtedly even better.

IEGULDĪJUMS TAVĀ NĀKOTNĒ

This work has been supported by the European Social Fund within the project «Support for Doctoral Studies at University of Latvia».

References

1. Wikiversity. [Online] [Quoted: 24 January, 2011] http://en.wikiversity.org/wiki/Software_testing/history_of_testing.
2. Bičevska, Z., Bičevskis, J.: Smart Technologies in Software Life Cycle. In: Münch, J., Abrahamsson, P. (eds.) Product-Focused Software Process Improvement. 8th International Conference, PROFES 2007, Riga, Latvia, July 2-4, 2007, LNCS, vol. 4589, pp. 262-272. Springer-Verlag, Berlin Heidelberg (2007).
3. Rauhvargers, K., Bicevskis, J.: Environment Testing Enabled Software - a Step Towards Execution Context Awareness. In: Hele-Mai Haav, Ahto Kalja (eds.) Databases and Information Systems, Selected Papers from the 8th International Baltic Conference, IOS Press vol. 187, pp. 169-179 (2009)
4. Rauhvargers, K.: On the Implementation of a Meta-data Driven Self Testing Model. In: Hruška, T., Madeyski, L., Ochodek, M. (eds.) Software Engineering Techniques in Progress, Brno, Czech Republic (2008).
5. Bičevska, Z., Bičevskis, J.: Applying of smart technologies in software development: Automated version updating. In: Scientific Papers University of Latvia, Computer Science and Information Technologies, vol .733, ISSN 1407-2157, pp. 24-37 (2008)
6. Ceriņa-Bērziņa J.,Bičevskis J., Karnītis Ģ.: Information systems development based on visual Domain Specific Language BiLingva. In: Accepted for publication in the 4th IFIP TC2 Central and East European Conference on Software Engineering Techniques (CEE-SET 2009), Krakow, Poland, Oktober 12-14, 2009
7. Ganek, A. G., Corbi, T. A.: The dawning of the autonomic computing era. In: IBM Systems Journal, vol. 42, no. 1, pp. 5-18 (2003)
8. Sterritt, R., Bustard, D.: Towards an autonomic computing environment. In: 14th International Workshop on Database and Expert Systems Applications (DEXA 2003), 2003. Proceedings, pp. 694 - 698 (2003)
9. Lightstone, S.: Foundations of Autonomic Computing Development. In: Proceedings of the Fourth IEEE international Workshop on Engineering of Autonomic and Autonomous Systems, pp. 163-171 (2007)
10. Diebelis, E., Takeris, V., Bičevskis, J.: Self-testing - new approach to software quality assurance. In: Proceedings of the 13th East-European Conference on Advances in Databases and Information Systems (ADBIS 2009), pp. 62-77. Riga, Latvia, September 7-10, 2009.
11. Bičevska, Z., Bičevskis, J.: Applying Self-Testing: Advantages and Limitations. In: Hele-Mai Haav, Ahto Kalja (eds.) Databases and Information Systems, Selected Papers from the 8th International Baltic Conference, IOS Press vol. 187, pp. 192-202 (2009).
12. Diebelis, E., Bičevskis, J.: An Implementation of Self-Testing. In: Proceedings of the 9th International Baltic Conference on Databases and Information Systems (Baltic DB&IS 2010), pp. 487-502. Riga, Latvia, July 5-7, 2010.
13. Diebelis, E., Bicevskis, J.: Test Points in Self-Testing. In: Marite Kirikova, Janis Barzdins (eds.)

Databases and Information Systems VI, Selected Papers from the Ninth International Baltic Conference. IOS Press vol. 224, pp. 309-321 (2011).
14. Automated Software Testing Magazine. [Online] [Quoted: 20 January, 2011] http://www.automatedtestinginstitute.com/home/index.php?option=com_conten t&view=article&id=1276&Itemid=122.
15. Automated Testing Institute. [Online] [Quoted: 15 January, 2011] http://www.automatedtestinginstitute.com.
16. Verify/ATI Conference. [Online] [Quoted: 2011. gada 12. maijā.] http://www.verifyati.com.
17. ATI Automation Honors. [Online] [Quoted: 20 January, 2011] http://www.automatedtestinginstitute.com/home/index.php?option=com_conten t&view=article&id=1262&Itemid=131.
18. ATI Schedule & Selection Process. [Online] [Quoted: 16 January, 2011] http://www.automatedtestinginstitute.com/home/index.php?option=com_content&view=article&id=1283&Itemid=156.
19. SmartBear Software. [Online] [Quoted: 15 December, 2010] http://www.automatedqa.com/products/testcomplete/.
20. Testing Geek - FitNesse Introduction. [Online] [Quoted: 3 April, 2011] http://www.testinggeek.com/index.php/testing-tools/test-execution/95-fitnesse-introduction.
21. Wikipedia - Wiki. [Online] [Quoted: 10 April, 2011] http://en.wikipedia.org/wiki/Wiki.
22. Wikipedia. [Online] [Quoted: 10 April, 2011] http://en.wikipedia.org/wiki/FitNesse.
23. T-Plan Robot Enterprise 2.0 Tutorial. [Online] [Quoted: 17 April, 2011] http://www.t-plan.com/robot/docs/tutorials/v2.0/index.html.
24. IBM Help System. [Online] [Quoted: 19 April, 2011] http://publib.boulder.ibm.com/infocenter/rfthelp/v7r0m0/index.jsp?topic=/com.ibm.rational.test.ft.proxysdk.doc/topics/c_pr_architecture.html.
25. HP - Looking for Mercury Interactive? [Online] [Quoted: 9 May, 2011] https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-10^36653_4000_100__&jumpid=reg_R1002_USEN#.
26. HP Unified Functional Testing software. [Online] [Quoted: 6 May, 2011] https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-127-24^1352_4000_100__.
27. Selenium Web Application Testing System. [Online] [Quoted: 30 April, 2011] http://seleniumhq.org/.
28. Pressman Roger, S. Software Engineering - A Practioner's approach 5th edition. New York : The McGraw-Hill Companies, Inc., 2001.
29. T-Plan Robot - Image Based vs Object Oriented Testing. [Online] [Quoted: 3 April, 2011] http://www.t-plan.com/robot/docs/articles/img_based_testing.html
30. Automatic Testing Frameworks. [Online] [Quoted: 19 April, 2011] http://www.automatictestingframeworks.com/?q=test_automation_framework.

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2012. Vol. 787
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 50–78 P.

Data Acquisition from Real World Objects based on Nonuniform Signal Sampling and Processing

Kaspars Sudars

University of Latvia, Faculty of Computing,
Institute of Electronics and Computer Science
sudars@edi.lv

The paper summarizes obtained research and practical results reported in the author's doctoral Thesis "Data Acquisition from Real World Objects". Work is focused on methods and algorithms for data acquisition from real world objects, and it is based on the theory of non-traditional Digital Signal Processing, including non-uniform sampling and pseudo-randomized quantizing. That leads to obtaining data simultaneously from an increased number of data sources, to widening the frequency range and to significant complexity reductions. In particular, research has been done in the directions of: (1) asymmetric data compression/reconstruction; (2) data acquisition from sources in the GHz frequency range; (3) rational acquisition of impedance data; (4) data acquisition based on signal and sine wave reference function crossings. Microelectronic implementation of the research results is considered. In particular, FPGA implementation of a Fast-DFT processor has been developed and tested.

Keywords. Data acquisition, Nonuniform signal sampling, Impedance measurements, DASP

1. Introduction

Motivation for research in the area of data acquisition for computer systems

Massive data acquisition from real life objects and supplying computers with this information in an effective way evidently is vital to realizing full potential of computer system applications in many areas. Various types of sensors are used for obtaining information from objects of natural or technical origin. Most of them convert the primary information into continuous-time or analog signals. Data acquisition (DAQ) from only this type of sensors is considered. As the acquired data are to be given as discrete quantities, the analog sensor signals have to be digitized to obtain their representations in the digital domain. Usually the classic Digital Signal Processing (DSP) technology is used for this.

This approach, based on the assumption that digitizing is based on the classical sampling and quantizing concepts, leads to significant narrowing of the digital domain, to using of more expensive analog signal processing methods and technical means for performing signal processing in the high and ultra high frequency range. It means that this approach negatively impact data acquisition technologies and limits their application range. The research described in this summary is motivated by the importance of achieving progress in the direction of computer system applications in the wide area of Information Technologies related to computer system interaction with real world biological and industrial objects.

Research goal and the basic problems that have to be resolved

Research activities of work target reaching the goal of discovering innovative methods for massive data acquisition from real life objects and effective supplying computers with this information. That evidently is vital for realizing full potential of computer system applications in many areas. The research is focused on resolution of the following basic problems that have been indicated at the beginning as the most essential:

- Too high DAQ system complexity;
- Relatively small quantity of sensors that typically can be connected to inputs of a single DAQ system;
- Limited number of channels for simultaneous data acquisition in parallel;
- Power consumption of DAQ systems, often limiting the duration of their autonomous performance time.

Basic tasks

To achieve progress in these directions, the following tasks are addressed:

1. Research focused on development of innovative methods and algorithms for complexity-reduced DAQ paying attention, in particular, on the following:
    - Combining data acquisition with signal specific digital pre-processing;
    - Reduction of power consumption;
    - Increasing the number of sensors that can be connected to a single DAQ system at least up to 100;
    - Compression of acquired data.
2. Development of algorithms and computer programs for sensor data transfer to computers.
3. Experimental investigations of the developed problem solutions, mostly by computer simulations in MATLAB environment.
4. Description of the developed DAQ structures in VHDL.

Approach to resolution of the considered problems

To reach the goal a flexible approach to complexity-reduced multi-channel data acquisition from a large quantity of sensors has been used. The developed methods, systems and algorithms for data acquisition from wideband, event timing and large distributed clusters of signal sources are discussed with emphasis on data gathering from a large quantity of signal sources. Special signal digitizing techniques, including pseudo-randomized multiplexing, time-to-digital conversions and signal sample value taking at time instants when the input signal crosses a sinusoidal reference function, are used for that. Development of the discussed massive data acquisition systems is based on the knowledge accumulated over a long period of time in the area of Digital Alias-free Signal Processing.

2. DAQ Systems for Information Gathering and Supplying to Computers

Relatively many various DAQ systems are currently produced and offered by many companies. On the other hand most of them actually are operating on the basis of a few basic DAQ principles that can be considered as classic as they have remained unchanged for a long time. Therefore performance of the existing DAQ systems mainly depends on the currently achieved perfection of the involved microelectronic elements.

Analysis of the currently used data acquisition system specifics reveals some essential facts and leads to the following generalized conclusions:

1. DAQ systems mostly belong to one of two basic types of DAQ systems: (1) multi-channel systems for DAQ based on multiplexing inputs to a central ADC connected via interface to a computer; (2) multi-channel systems for so-called simultaneous DAQ using a separate ADC in every input channel.
2. DAQ functions usually are considered as input signal conditioning, analog-to-digital conversions and transmitting the digital signals, obtained from all inputs, to the computer. Processing of data more often than not is the responsibility of the host computer.
3. The essential function of signal digitizing usually is performed on the basis of the classical concepts of uniform periodic sampling (Shannon sampling theorem) and fixed-threshold quantizing.
4. The sampling rate of for DAQ systems based on multiplexing inputs to a central ADC directly depends on the clock frequency at which the multiplexer is switched and the rate of sampling signals in each input channel is inversely proportional to the number of inputs.
5. Consequently the quantity of DAQ system inputs usually is restricted to relatively small numbers, usually up to 16. More complicated systems con-

tain a hierarchy of multiplexers. In this case there might be more inputs however this type of DAQ systems then can be used only for data acquisition from low-frequency sources.

6. Using a separate ADC in every input channel allows avoiding these restrictions. Therefore this type of DAQ systems can be used for obtaining data from high frequency signal sources. The factors limiting their applications are relatively high complexity (ADC in every channel) and multiplexing of ADC output signals needed for transmitting them sequentially to the host computer. The quality of developed and produced DAQ systems improve all the time, however this progress is mostly based on the achievements in the area of semiconductor device development and production technologies.

The last conclusion is true for most of the currently produced DAQ equipment. On the other hand, there have been various R&D efforts addressing the problem of data acquisition under conditions more demanding than usual.

In particular, the applicability of NU sampling and low-bit rate quantizing has been investigated over a long period of time and this approach is directly related to DAQ functions [15, 16, 33, 34]. These methods and algorithms are applicable for achieving a number of essential advantages, such as performing of DAQ in a wide frequency range, elimination the dependency of the sampling rate in a channel on the quantity of channels in a system, data compression/reconstruction, reducing the complexity of DAQ systems and others. That leads to the following conclusions:

1. The currently used classical theory covering signal digitizing (sampling and quantizing) and digital representation of analog signals is not exclusive. These signal conversion operations can be performed in various ways, based on the signal digitizing theory developed in a few past decades, including theory of randomized signal processing, NU sampling and quantizing and DASP.
2. To achieve progress in the area of DAQ, efforts have to be put in this work basically in the directions of developing methods and algorithms for application-specific DAQ.
3. Selecting and using the most effective type of digital representation of analog input signals is really important for that.
4. Data pre-processing in many cases should be included in the list of functions to be fulfilled by a DAQ system as algorithms for parallel processing of raw digital signals can be developed and used at that stage for effective data representation and compression.

Reduction of power consumption, simplification of system hardware, simplification of algorithms for acquired data pre-processing and enlargement of system channel quantity, accurate signal acquisition are still very desirable for DAQ systems.

3. Data Acquisition Based on NU Sampling: Achievable Advantages and Involved Problems

Periodic sampling is currently the most used and widespread sampling method. However there are other methods for sampling, specifically, NU sampling. The achievable advantages and application potential of this approach are based on exploitation of the capabilities offered by the NU sampling procedure carried out in the process of the sensor signal digitizing [15, 16, 23, 33, 34, 38].

Whenever continuous time signals are digitized and then processed on the basis of DSP, the sampling rate f_s of the used periodic sampling limits the bandwidth of the original analog signals. Then the restrictions, defined by the Sampling theorem, have to be satisfied to avoid the uncertainty due to the fact that all frequencies belonging to the sequence: $f_o; f_s \pm f_o; 2f_s \pm f_o; 3f_s \pm f_o; \ldots nf_s \pm f_o$ are indistinguishable.

However that is true only under the conditions of periodic sampling. The situation is different whenever DAQ is based on the lately developed signal digitizing theory, including theory of randomized signal processing, NU sampling and quantizing and DASP. Specifically, the basic effect achieved by using NU sampling is avoiding aliasing. This achievable capability is very important for DAQ.

Problems related to NU based DAQ are investigated. The most significant is the problem of cross-interference (CI) between signal spectral components occurring whenever the signal sample values are taken at non-equidistant time instants. Impact of CI on DAQ is taken into account and CI is suppressed.

On the other hand, the obtained results show that various specific advantages and benefits, valuable for practical applications, can be obtained by exploiting NU sampling techniques. Several methods of this kind are considered further.

Data acquisition from high frequency signal sources

As proper application of NU sampling leads to elimination of aliasing, this specific approach to signal sampling can be used to enlarge the frequency range where information carried by high frequency analog sensor signals can be represented by digital data (for example, in cases of signal demodulations).

A structure of DAQ systems based on NU is considered. It can be used for data acquisition directly from high frequency signal sources if operation of the used ADCs is based on quantizing signal sample values taken from the input signal at time instants dictated by a NU sampling point process. In such a case the usage of the additive sampling point process is usually preferable. The upper frequency of the signal spectrum sometimes can exceed the mean sampling rate several times. This makes it possible to acquire data in a wide frequency range extending up to few GHz. However the data obtained under these conditions then must be treated with the specifics of NU sampling taken into account [12, 13, 17].

Increasing the quantity of input channels

The structure of the DAQ system used for data acquisition from a multitude of sensors is based on multiplexing of analog signals. The difference between the traditional structures of this type and this one is in operation of the multiplexer. Whenever it connects inputs to the ADC periodically, the number of inputs is limited by the achievable multiplexer switching rate as the sampling rate of each input then is n times lower, where n is the number of inputs. Replacing the periodic multiplexer switching by randomized allows using the total bandwidth of channels more efficiently. Therefore that makes it possible to increase the number of inputs several times. [14].

Data acquisition at object testing

Exploiting NU sampling techniques and gaining advantages on that basis is a considerably less complicated task under conditions typical for the cases where data are acquired from some biomedical or industrial objects that are being tested. Then the information that has to be obtained is related to the characteristics and properties of the object and the signals taken off the respective object during the tests carry it. These signals reflect reaction of the object on some specific excitation signals that are used. The fact that data acquisition then is performed under conditions where the test signal characteristics, including its spectrum, are given is very useful. Taking this information into account makes it possible to reduce the complexity of the following data processing process significantly. Using of this a priori information leads to substantial simplification of the algorithms. All cross-interference coefficients then can be pre-calculated, matrix C of their values also can be composed and inverted so that all elements $\square_{ij}$ of the inverted matrix inv(C) also could be pre-calculated. Therefore the Fourier coefficients in this case can be calculated directly. That dramatically simplifies and speeds-up the involved calculations.

Using of this approach for data acquisition in the specific case of bioimpedance signal analysis in a wide frequency range of the test signals (up to a few GHz) is described. [35].

Fault tolerant data acquisition

Suppose data are acquired under conditions where the spectra of the involved processes are restricted at relatively low frequencies. Then application of NU sampling is not needed and data might and should be acquired at periodically repeating time instants. Attention is drawn to the possibility of improving the performance of the data acquisition system on the basis of NU sampling even under these conditions if it is needed to protect this system against unpredictable explosive noise bursts. Then the concept of NU sampling and specific data processing typical for NU sampling still proves to be quite useful.

Time diagrams shown in Figure 1 illustrate the data acquisition process under normal conditions in Figure 1(a) and under the impact of noise bursts in Figure 1(b).

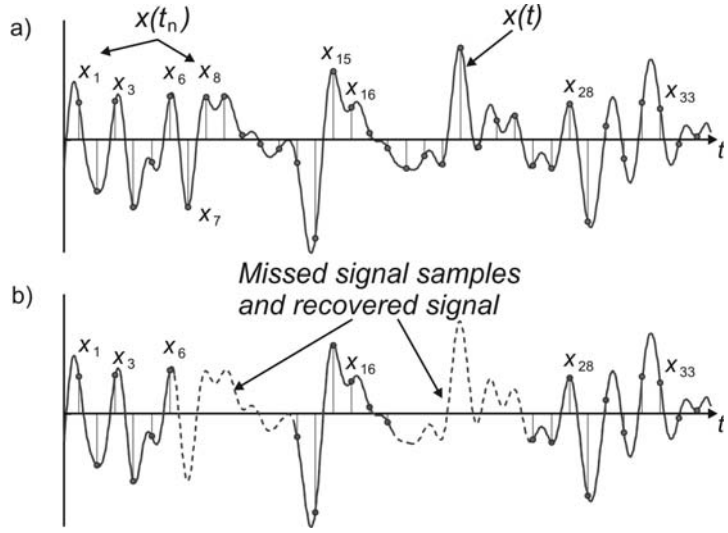


Figure 1. Impact of noise bursts on signals: (a) periodically sampled continuous signal, (b) the same signal with bursts of lost data and the recovered parts of faulted signal.

The idea of improving fault tolerance on this basis is simple. The essence of it is as follows: to improve fault tolerance on the basis of the NU sampling concept, the data should be acquired periodically as usual and facilities usually used for processing of nonuniformly acquired data should be added to the system and used for data reconstruction [12, 13, 17]. This means that under normal data acquisition conditions redundant equipment would be used. The role of this equipment would be to recover the data lost under impact of the noise bursts.

Asymmetric data compression/reconstruction

Data compression function plays a significant role at DAQ. Essential are the advantages related to data compression obtainable if proper NU sampling procedures are used.

At standard data compression/reconstruction data are processed twice to compress and decompress them. Naturally that requires using of computing resources twice and that takes time. Using of the NU sampling procedures makes it possible to reduce the volume of data representing the respective input sensor signals simply by taking out some quantity of the signal sample values or perform other simple operation. It means that in this case no calculations are made at data compression. The computational burden related to reconstruction of the compressed

data then is totally placed on the data recovery side of the system. This data acquisition paradigm seems to be with high application potential as it is well suited for significant reducing of the acquired data massive as well as the data compressing costs in terms of equipment volume, weight and power consumption. The basic advantage is that this type of complexity-reduced data acquisition and compression is fast [15]. Note that compressive sensing also belongs to asymmetric data compression/reconstruction paradigm.

Asymmetric 2D image data acquisition/reconstruction

The considered method for asymmetric data acquisition/reconstruction can be used in a wide application area, including 2D data compression and reconstruction [10]. Therefore this method can be exploited for image encoding, transmission (or storage) and reconstruction. Actually this task is quite complicated and can be done using different reconstruction methods. One possible example is considered further and the standard test image, shown in Figure 2(a), is used for that. This standard test image I can be defined by its $M \times N$ matrix I of pixels or elements as follows:

$$I = \begin{bmatrix} e_{11} & e_{12} & e_{13} & e_{14} & e_{15} & e_{16} & \dots & e_{1N} \\ e_{21} & e_{22} & e_{23} & e_{24} & e_{25} & e_{26} & \dots & e_{2N} \\ e_{31} & e_{32} & e_{33} & e_{34} & e_{35} & e_{36} & \dots & e_{3N} \\ e_{41} & e_{42} & e_{43} & e_{44} & e_{45} & e_{46} & \dots & e_{4N} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ e_{M1} & e_{M2} & e_{M3} & e_{M4} & e_{M5} & e_{M6} & \dots & e_{MN} \end{bmatrix} \quad (1)$$

According to the considered compression method, many of pixels have to be replaced by zeroes using NU signal sampling. The best of so far found approaches are based on design and usage of a mask containing logic 1 and 0. An example of it is given as matrix H.

$$H = \begin{bmatrix} 0\,1\,0\,0\,0\,1\,0\,0\,1\,0\,0\,1\,0\,0 \dots 0 \\ 0\,0\,1\,0\,0\,0\,1\,0\,0\,1\,0\,0\,1\,0 \dots 0 \\ 0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,1\,0\,0\,1 \dots 1 \\ 1\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,1\,0\,0 \dots 0 \\ 0\,1\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,1\,0 \dots 0 \\ \dots \ \dots \ \dots \ \dots \ \dots \ \dots \ \dots \ \dots \\ 0\,1\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0\,1\,0 \dots 0 \end{bmatrix} \quad (2)$$

To generate this mask, taking out of pixels is done in a specific way by using additive random sampling sequence. The mean distance μ of two neighbouring pixels left in the image is equal to 5 and width of the uniform distribution of pseudo-random numbers $\square$ is equal to 3.

The compressed image is obtained in a very simple and fast way, just by using logic or scalar multiplication of generated mask matrix with the image matrix:

$$I_c = I \times H = \begin{bmatrix} 0 & e_{12} & 0 & 0 & 0 & e_{16} & 0 & 0 & e_{19} \ldots & e_{1N} \\ 0 & 0 & e_{23} & 0 & 0 & 0 & e_{27} & 0 & 0 \ldots & e_{2N} \\ 0 & 0 & 0 & e_{34} & 0 & 0 & 0 & e_{38} & 0 \ldots & e_{3N} \\ e_{41} & 0 & 0 & 0 & e_{45} & 0 & 0 & 0 & e_{49} \ldots & e_{4N} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \\ 0 & e_{M2} & 0 & 0 & e_{M5} & 0 & 0 & 0 & e_{M9} \ldots & e_{MN} \end{bmatrix} \quad (3)$$

The obtained sparse image after losing 80.27% of its pixels is shown in Figure 2(b). Evidently image compression performed in such a way is a quite inexpensive and technically effective operation. The complexity of this type of image compression is much lower than the complexity of the usually exploited rather complicated standard image compression algorithms. That apparently represents a significant advantage of the proposed and described 2D data compression algorithm.

Reconstruction of the compressed image

Two approaches of sparse image reconstruction were considered and studied. Both of them have a common first recovery stage. The result obtained after this stage is shown in Figure 2(c). According to the suggested reconstruction method, part of all unknown image pixels can be calculated at this stage by processing its subsequent pixel values if these are known. In this particular sampling case, the maximum count of the known pixels in close proximity (considering 2D 4-pixel connectivity case) can be 2. These pixels can be approximately calculated as follows:

$$\begin{aligned} \hat{e}_{mn,l} &= 0.5(e_{m-1,n} + e_{m,n+1}), \ \textit{for the left side estimates} \\ \hat{e}_{mn,r} &= 0.5(e_{m+1,n} + e_{m,n-1}), \ \textit{for the right side estimates} \end{aligned} \quad (4)$$

Where $m = 1, 2, 3 \ldots M$, $n = 1, 2, 3 \ldots N$. The matrix of the recovered image pixel values after the first reconstruction cycle is the following:

$$\hat{I}_{r1} = \begin{bmatrix} \hat{e}_{11} & e_{12} & \hat{e}_{13} & 0 & \hat{e}_{15} & e_{16} & \hat{e}_{17} & 0 & e_{19} \ldots & e_{1N} \\ 0 & \hat{e}_{22} & e_{23} & \hat{e}_{24} & 0 & \hat{e}_{26} & e_{27} & \hat{e}_{28} & 0 \ldots & e_{2N} \\ 0 & 0 & \hat{e}_{33} & e_{34} & \hat{e}_{35} & 0 & \hat{e}_{37} & e_{38} & \hat{e}_{39} \ldots & e_{3N} \\ e_{41} & \hat{e}_{42} & 0 & \hat{e}_{44} & e_{45} & \hat{e}_{46} & 0 & \hat{e}_{48} & e_{49} \ldots & e_{4N} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \hat{e}_{M1} & e_{M2} & 0 & \hat{e}_{M4} & e_{M5} & 0 & 0 & \hat{e}_{M8} & e_{M9} \ldots & e_{MN} \end{bmatrix} \quad (5)$$

At the next stages this process can be continued iteratively. All the pixel values are estimated in this way after a few cycles and the full image is recovered.

The average relative error, obtained in this particular case at image reconstruction performed in accordance to the described method, is equal to 0.1648 %.

The second considered reconstruction method is based on application of SECOEX method. That leads to the result shown in Figure 2(e). In this particular case, the average relative error of image recovery is 0.3147 %, what actually is worse than the result obtained by the previous method. While this larger error can be explained by the particularities of the processed test image signals, image reconstruction based on the SECOEX method is much more complicated and more time consuming in comparison with the developed method.

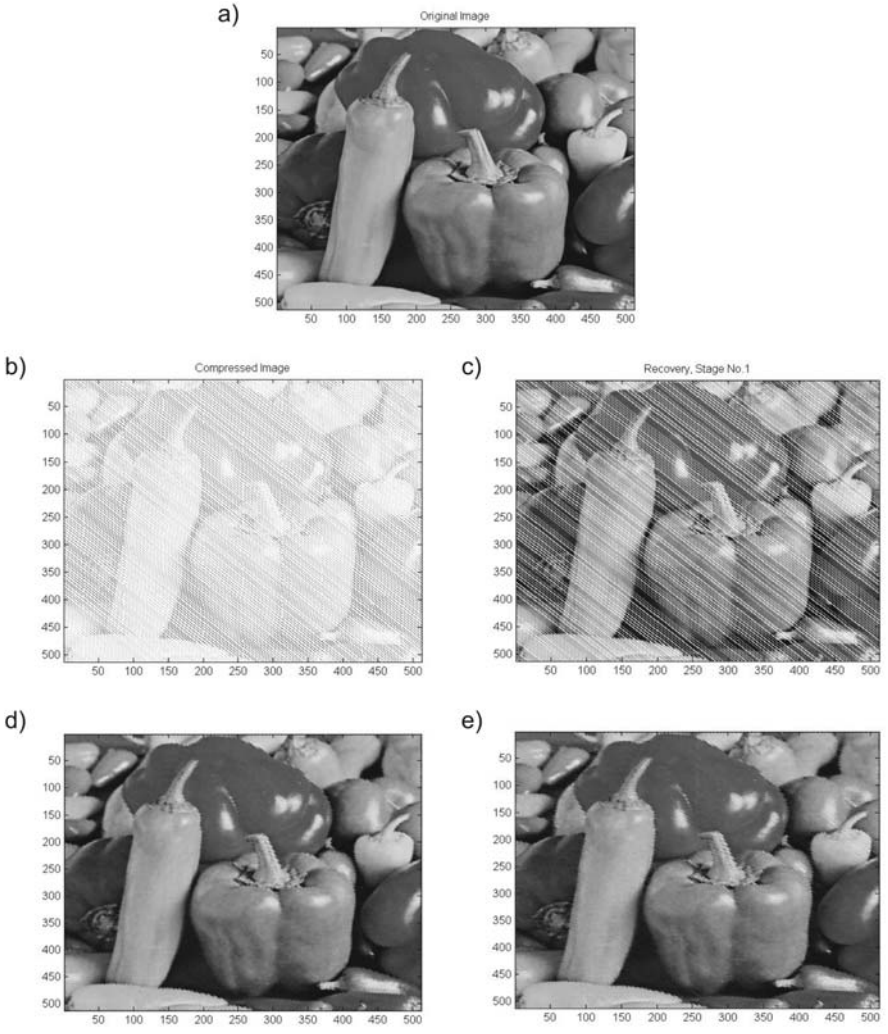


Figure 2. Example of asymmetric DAQ compression/reconstruction: (a) the original image by size 512×512 pixels; (b) sparse image after losing 80.27% of its pixels; (c) the image recovered at the first recovery stage; (d) image recovered by calculating unknown pixels from their neighbours and (e) recovered image obtained on the basis of SECOEX method.

The asymmetric 2D data acquisition/reconstruction has significant advantages for application cases where computational power is limited and transmitting of the acquired data is expensive in terms of spent time, memory and equipment resources.

The applicability of this method has been widened to cover also processing of colour images. An experimental system has been developed and made for studies of the described image compression method.

Wideband signal digitizing in enlarged dynamic range

The frequency range, where the currently available 10 to 12 bit ADCs are applicable, often is not wide enough. On the other hand, the dynamic range of the ADCs, applicable for analog-digital conversions in GHz frequency range, is limited by the achievable quantization bit rate usually not exceeding 4 bits [19, 41]. These typical problems, arising at attempts to convert analog signals into their digital counterparts in a wide frequency range extending up to GHz frequencies, are studied and specific approaches to resolution of them is suggested further.

Combining precise and low-bit very fast sampling

Combining precise and low-bit very fast sampling, in particular, significantly widen the area where DAQ can be performed in GHz frequency range by using low bit rate ADC. This type of signal sampling/reconstruction scheme is shown in Figure 3. The sampling operation in this case is based on adding to the precise ADC another low-bit rate one, which has to be very fast.

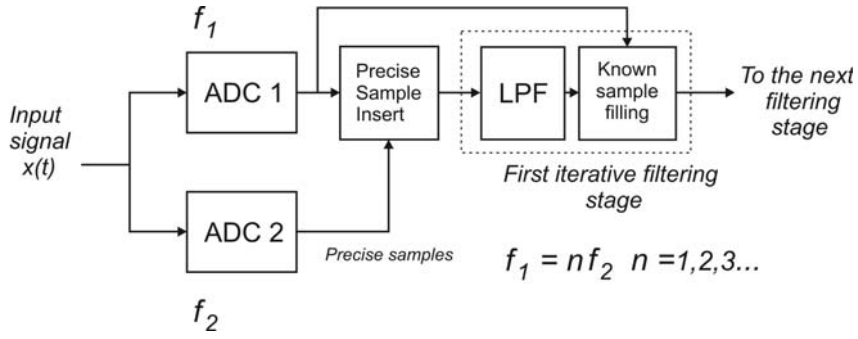


Figure 3. Structure of the electronic systems used for iterative reconstruction of signal waveforms, when signal is sampled using two ACDs in parallel.

Reconstruction of signal waveforms in this case is based on iterative filtering procedure. Results of 4 bit fast periodic sampling are used for filling the spaces remaining empty after the precise signal sample values have been taken at time instants according to the described additive random sampling. Then the conditions for waveform reconstruction are more favourable and the obtained results also are

significantly improved. They are displayed in Figure 4. To further improve the waveform reconstruction, using of yet another one of the randomized procedures, namely, randomized quantizing is suggested and the results obtained in that case also are shown there.

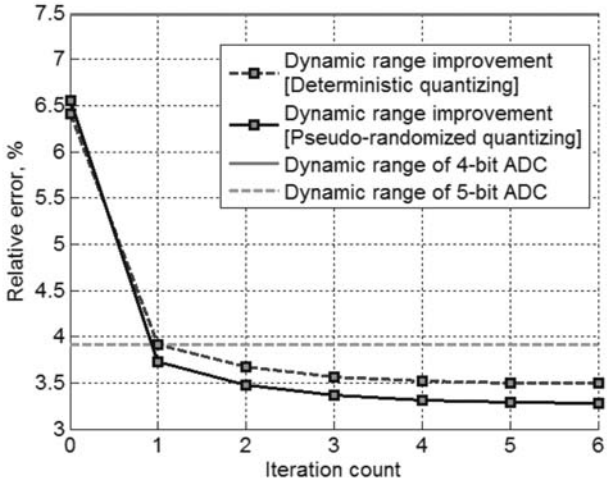


Figure 4. Results of waveform reconstruction when precise undersampling and low-bit fast periodic sampling procedures are used in parallel. Using of pseudo-random quantizing improves the precision.

Conclusions

A number of benefits achievable for DAQ by using the concept of NU sampling are described to show the application potential of DASP methods for data acquisition. Considered approaches are beneficial especially for:

- DAQ based on pseudo-randomized multiplexing;
- Asymmetric data compression/reconstruction;
- Data acquisition from objects under tests;
- Improving fault tolerance of data acquisition.

NU sampling has high potential in the field of 2D data acquisition or image data acquisition. To show this, the example of asymmetric image data compression/reconstruction was considered. The standard test image has been taken and compressed taking out 80% of its pixels according to the asymmetric DAQ concept. After that it was successfully reconstructed with 0.1648 % of average relative error. Thus it has been shown that application of NU sampling makes it possible to perform very simple image data compression. The achieved compression rate, in the case of the considered example, is equal to 5. It is shown that the developed method and algorithm can be used also for 2D data acquisition for colour image data compression and reconstruction with the same compression rate of 5.

5. Methods for Data Acquisition Exploiting Advantages of Pseudo-randomized Quantizing

Specifics of DAQ based on pseudo-randomized quantizing (PRQ) are considered in more detail in the particular case of impedance measurement data acquisition and pre-processing. Impact of signal quantizing on DAQ resolution and DFT coefficient estimation precision is considered and the potential of pseudo-randomized quantizing for resolution improvement is shown.

The impedance (including bioimpedance) measurements in many cases could be reduced to estimation of the signal spectra on specific pre-determined frequencies. The task of this spectrum analysis is made more difficult by the fact that the frequency range of interest is wide. While effective methods and techniques for impedance measurements at frequencies up to several MHz have been developed and are used, there are problems when the spectrum analysis of the modulated impedance signals has to be performed in the frequency range up to several hundreds of MHz or even up to several GHz. The involved data acquisition and processing tasks then become rather challenging. In particular, data have to be processed in real-time and with sufficiently high resolution.

Processing of impedance signals on the basis of complexity reduced DFT

The pseudo-randomly quantized signal values contain significantly increased number of bits in comparison with deterministically or randomly quantized signals as the value of the pseudo-random noise $\square_k$, used at quantizing, is added [15, 16]. Directly processing of the quantized signal clearly leads to a complicated processing procedure. The problem was considered and it is shown how to avoid these complications. Suppose impedance signal demodulation is on estimation of the Fourier coefficients.

It is suggested to do that on the basis of the following equations:

$$\begin{aligned}\hat{a}_i &= \frac{2}{N}\sum_{k=1}^{N}\hat{x}_k \cos 2\pi f_i t_k = \frac{2q}{N}\sum_{k=1}^{N} n_k \cos 2\pi f_i t_k + \gamma_i^a \\ \hat{b}_i &= \frac{2}{N}\sum_{k=1}^{N}\hat{x}_k \sin 2\pi f_i t_k = \frac{2q}{N}\sum_{k=1}^{N} n_k \sin 2\pi f_i t_k + \gamma_i^b\end{aligned} \tag{6}$$

Where γ_i^a and γ_i^b are corrections:

$$\begin{aligned}\gamma_i^a &= \frac{2q}{N}\sum_{k=1}^{N}(\xi_k - \frac{1}{2})\cos 2\pi f_i t_k \\ \gamma_i^b &= \frac{2q}{N}\sum_{k=1}^{N}(\xi_k - \frac{1}{2})\sin 2\pi f_i t_k\end{aligned} \tag{7}$$

These corrections remain constant, can be pre-calculated for all excitation frequencies and used. The advantage of this approach to estimation of the Fourier coefficients is evident. Data can be processed in this case in a very simple and fast way.

Obtained results of the suggested method

Diagrams given in Figure 5 illustrate the precision obtainable at using the suggested methods for impedance data acquisition and demodulation of the impedance signals.

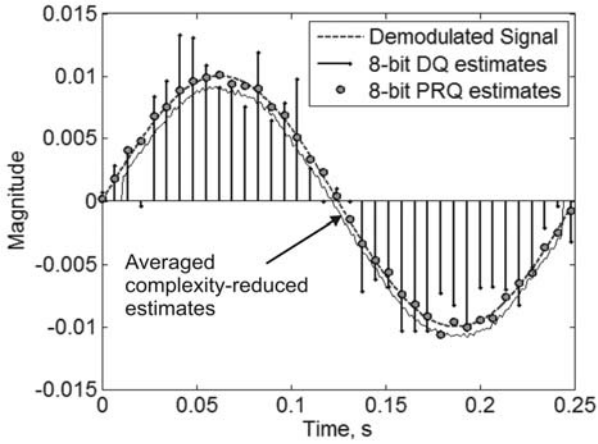


Figure 5. Estimates of the demodulated signal obtained by using deterministic quantizing (1) and pseudo-randomized quantizing (2). Averaged estimates based on the complexity-reduced approach using sign(sin) and sign(cos) functions are given as curve (3).

Demodulation quality improvement by filtering

After impedance signal demodulation, the extra precision of processing could be obtained with ordinary digital filtering. For this purpose application of the moving average filter MAF is considered. If it is needed such filtering could be used iteratively. Also it is expected that usage of other filter types, like low-pass filters LPF, would provide good results.

Figures in Table 1 illustrate comparison of signal quantization errors with and without filtering. As can be seen, the filtering in general reduces quantizing errors, no matter whether it is MAF or LPF filtering.

Table 1

Typical relative quantizing error readings.

	Moving average filter MAF length	3	5	7	9
1	Deterministic 4 bit quantizing (no filtering)	0.1010	0.1010	0.1010	0.1010
2	Deterministic 12 bit quantizing (no filtering)	0.00037	0.00037	0.00037	0.00037
3	4+4 bit PRQ quantizing (no filtering)	0.0994	0.0993	0.1005	0.1004
4	4+4 bit PRQ quantizing, LPF	0.0431	0.0403	0.0426	0.0406
5	Deterministic 4 bit quantizing, LPF	0.0642	0.0642	0.0642	0.0642
6	4+4 bit PRQ quantizing, MAF	0.0399	0.0294	0.0274	0.0271

As it can be seen the PRQ quantizing provides significantly better demodulated signal precision than the traditional deterministic quantizing.

Conclusions

Application specifics of the basic three signal quantizing methods (methods for deterministic, randomized and pseudo-randomized quantizing) for data acquisition are studied. In many cases PRQ proves to be the best of them. That, of course, depends on the particular application.

DFT is often needed for biomedical DAQ applications, where performance of bioimpedance signal demodulation is required. Methods for such signal processing are essential for design of medical equipment. This work considers impedance signal demodulation based on calculations of DFT coefficients on several frequencies and the advantages of using PRQ are shown.

Due to these features, PRQ is an approach to quantizing that is rather valuable for achieving high performance of DAQ systems. That is confirmed by results obtained in the area of DAQ used for impedance data acquisition.

It is shown that in this application area:

- Improved resolution of very fast low bit rate ADCs (several GHz sampling frequency) at wideband low bit rate DAQ. In other words, PRQ reduces the quantizing bit rate, widening the application range of rough quantizing;
- Better digitized signal quality (in terms of signal sample values precision);
- Simplified, energy efficient DAQ hardware and impedance signal processing.

In general, the obtained research results confirm that design and performance of DAQ systems often can be significantly improved by exploiting PRQ.

5. Data Acquisition Based on Gathering Timing Information

The specific approach to signal sampling, based on detection of signal and reference function crossings is considered further. It is shown that under certain conditions it is possible to represent analog signals with timed sequences of events without loss of information and that this possibility can be exploited for developing competitive DAQ systems based on detection of signal and reference crossing events (SRC sampling) that have attractive and useful advantages.

The comprehensive comparison of the classical signal sampling method and suggested method for SRC sampling is given in Table 2.

Table 2

Comparison of two alternative sampling approaches.

Classical sampling approach	SRC sampling approach
Signal sample value acquiring	
• Signal sample values are taken at predetermined time instants • Sample values are represented by respective voltage (current) levels • Uniformly spaced signal sample values • Remote sampling implementations usually not acceptable • Aliasing determined by Nyquist frequency	• Sample values are taken at the signal and reference function crossing instants • Sample values are represented by the time instants when the crossing events take place • Nonuniformly spaced signal sample values • Remote sampling applicable • Specific aliasing conditions • Reference function defines the envelope of the sampled signal instantaneous values
Signal sample value transmission	
• Requires transmission of constant levels • Sensitive to the ambient noise • Transmission acceptable only over short distances	• Based on transmission of time instants • Relatively insensitive to the ambient noise • Transmission might be performed over relatively large distances • Provides for data compression
Quantizing	
• Sample values are quantized directly	• Reference function values are quantized at the crossing time instants • High precision synchronization is required
Multi-channel operation	
• Based on switching the analog input signals • Number of channels limited by hierarchic multiplexer structures	• No switching of analog signal performed • Multiplexing of channels, provided by enabling function, is much simpler • Well suited to data acquisition from a large quantity of signal sources • Well suited to the specifics of Ultra Wideband communication
Processing	
• Based on the classical DSP algorithms	• Typically specific algorithms are needed for processing • Applicability of standard algorithms achievable under certain condition • Complexity-reduced pre-processing
Advantages for DAQ systems	
• Features and obtainable benefits are well known	• Reduced complexity and power consumption at the DAQ system front-end part • Significantly enlarged number of data channels

Preference is given to the method for DAQ based on detection of signal and sine-wave crossing time instants, so-called SWC signal sampling method. The sine-wave is preferred as a reference function due to its positive characteristics, in particular, spectral purity of it. This significant feature of SWC sampling might be exploited for reducing the complexity of the algorithms for pre-processing of signals.

Potential of SWC sampling for simplifying some DSP algorithms

Algorithms for processing digital signals obtained in result of SWC sampling often can be significantly less complicated than the widely used conventional ones. In particular, in many cases it is possible to avoid multiplications of multi-bit numbers. The unique constant envelope feature of SWC sampling makes it possible to develop rational algorithms, in particular, for Discrete Fourier Transform based spectrum analysis and waveform reconstruction. For example, the following equations can be used for calculation of Fourier coefficient estimates $\hat{a}_i$, $\hat{b}_i$ not requiring multiplication of multi-digit numbers as usual [1, 2, 6]:

$$\begin{aligned}\hat{a}_i &= \frac{A_r}{N}\sum_{k=0}^{N-1}(\sin 2\pi(f_r - f_i)t_k + \sin 2\pi(f_r + f_i)t_k)\\ \hat{b}_i &= \frac{A_r}{N}\sum_{k=0}^{N-1}(\cos 2\pi(f_r - f_i)t_k - \cos 2\pi(f_r + f_i)t_k)\end{aligned} \tag{8}$$

Where A_r, f_r are the amplitude and frequency of the sinusoidal reference function, t_k denotes the crossing time instants and N is the number of signal samples processed.

This approach to complexity reduction of algorithms for data pre-processing is usable only under the condition that the reference function is sinusoidal. This represents a strong argument in favour of using this type of reference functions. The problem related to the necessity of operating with sine and cosine functions in this case can be easily resolved by using look-up tables. The developed method for complexity-reduced digital filtering and parameter estimation is considered as an invention and the respective European patent application is published [6].

SWC sampled signal properties in the frequency domain

SWC sampled signal properties in the frequency domain are quite specific and they were explored in detail. First of all, SWC represents a specific modification of NU sampling and, as it is typical for this type of sampling, SWC has anti-aliasing properties. It is shown that there is no full-scale frequency overlapping. While peaks appear in the spectrogram at some indicated and defined frequencies, they are significantly suppressed. This type of aliasing is considered as so-called fuzzy aliasing [15].

Data acquisition based on SWC sampling

Data acquisition based on SWC sampling is illustrated in Figure 6. Digitally timed events, defined as crossings of an input signal and a reference function, is used in this case for representation of analog signals in the digital domain. The most responsible elements of this system apparently are the comparators used for detection of the time instants t_k at which the signal *x(t)* intersects the reference function *r(t)* that is given as a sine wave at frequency f_r. It has constant amplitude A_r. The physical output *y(t)* of the comparator is formed as a pulse whenever the sampler is enabled by a function *z(t)* (Figure 6(b)) and a crossing of the input signal *x(t)* and the reference function *r(t)* takes place.

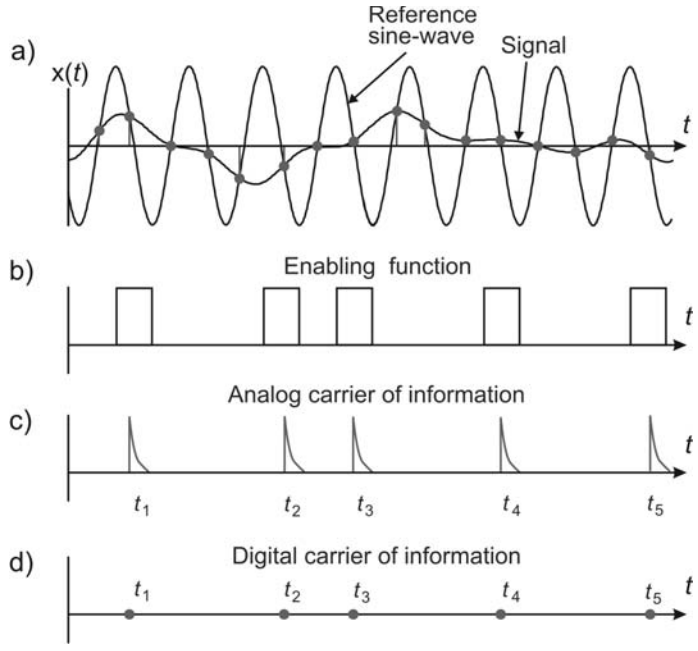


Figure 6. Time diagrams illustrating specific two types of input signal representation in the digital domain. (a) sampling based on signal and the reference function crossings; (b) enabling function; (c) analog carrier of information; (d) digital carrier of information.

These pulses *y(t)* at the sampler output are formed so that they carry the timing information indicating the exact crossing instants t_k of the signal and the reference function. It is assumed that the output pulses, having sharp front edges are formed with a constant delay after each signal and the reference sine-wave crossings. Thus the sampler output signal is carrying the original information encoded as the sequence $\{t_k\}$ of the sine-wave crossing instants. It is transferred over some shorter or longer distance to the host computer. Note that this sequence fully represents the respective input signal. It means that the crossing instant sequence might be

used either for recovery of the input signal sample values and reconstruction of the signal or the input data processing might be based on direct processing of these crossing instants as it is explained below [8]. While the sequence of the signal and reference function crossing instants represents in this case the input analog signal in the digital domain, either an analog or a digital carrier may be used for transmitting this information. A train of position-modulated short pulses (Figure 6(c)) is used as an analog information carrier and a sequence of digital crossing instant values $\{t_k\}$ (Figure 6(d)).

The analog carrier is used primarily for gathering and transmission of data from the cluster of remote samplers to the master part of the distributed ADC (Figure 7). The digital carrier is used at the stages of data reconstruction and/or their preprocessing and data transfer to computers.

While various techniques might be used for the recovery of the input signal sample values, basically a replica of the reference function waveform is sampled and quantized for that. Note that this copy of the reference function could be given either in analog or digital format. The signal sample values, of course, might be recovered and presented digitally in both cases.

Architecture of an energy-efficient system for multichannel data acquisition

The suggested hardware architecture for simultaneous multichannel data acquisition is shown in Figure 7. Reduction of power consumption to a large extent is due to the method used for sampling. Application of it not only leads to the simplicity of the front-end design of this system. Remote samplers that can be placed directly at the locations of the sensors then are used rather than ADCs. This approach is well suited for remote signal sampling applications and for building the shown architecture of the system that actually is a distributed ADC. As can be seen, the sampling and quantising operations, in this structure, are distanced. This approach makes it possible to use many remote samplers at the front-end of it and to gather data from them in a rational way. To ensure proper performance of them in parallel, specific enabling control functions are introduced and used. Only those crossings are taken into account that happen during the time intervals when the respective comparator is enabled by a specially generated enabling function. This enabling function is exploited also for executing the input multiplexing. The analog input signal switching could be avoided then and that certainly is a significant positive fact.

At the current technological level, if the SWC based DAQ is implemented on available ICs, this limit is approximately 25 *MHz* for 8 to 10 bit quantizing. It is not so easy to achieve it. More realistic figures are 5–10 *MHz* with the quantization rate up to 12 bits. This result was obtained testing the experimental system developed in the framework of ERAF project Nr. VDP1/ERAF/CFLA/05/APK/2.5.1/000024/012 "Development of multi-channel systems for acquisitions of data from biomedical,

ecological and industrial systems and transferring them to computerized systems", co-sponsored by the European Union. This leads to the maximal number of channels equal to 1000 for the mean sampling frequency equal to 20 *KHz*. Then the input signal could be processed in the alias-free way within the bandwidth 0–10 *KHz*. More often than not the required input numbers are smaller. Then the input signal bandwidth could be proportionally widened. To achieve results fully reflecting the potential of this approach, special ASICs for the analog-digital front end devices must be developed and used.

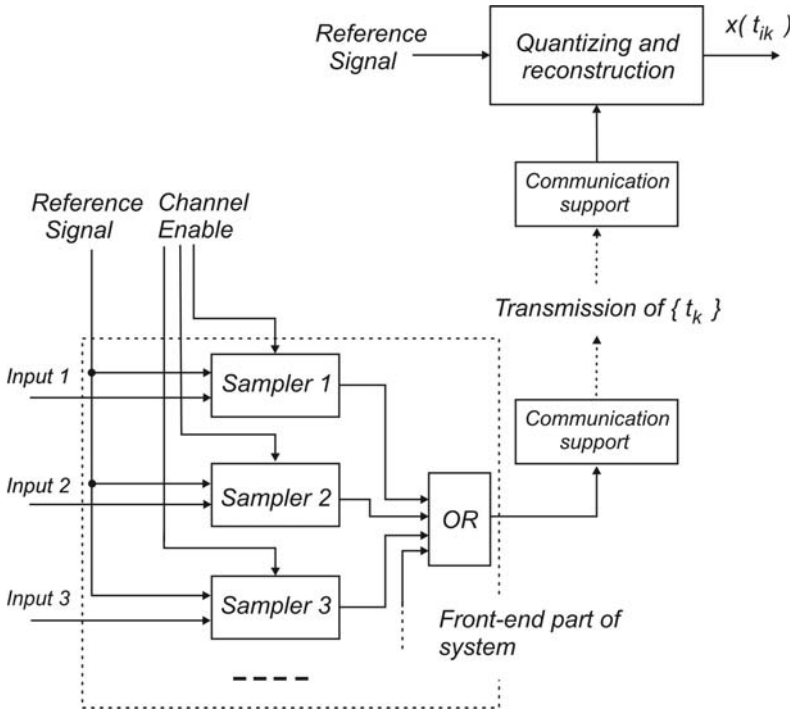


Figure 7. Architecture of the considered energy-efficient system fulfilling simultaneous data acquisition functions based on SWC sampling.

Achieving the applicability of standard DSP algorithms

To avoid SWC sampling nonuniformities, regularization of SWC sampling results is considered. The very simple method for this regularization has been developed and is suggested. It has remarkable advantages in comparison with other reconstruction methods where burdensome signal processing is involved. The suggested signal regularization does not require any additional computation power at all. Of course, the results may be acceptable for end-users only under certain conditions.

Regularization of SWC sampling leads to obtain the possibility of using standard DSP algorithms for processing SWC sampled signals. Whenever this type of regularization is carried out, the wealth of existing DSP algorithms can be used for processing the signal sample values obtained in result of SWC sampling. This method can be implemented with help of the enabling function.

The curves in Figure 8 have been obtained varying signal sampling frequency and keeping constant reference frequency. The reference frequency, with the enabling decimation factor n varying, was changed according to $f_s = f_r/n$. Then with n growing, the sampling conditions are becoming closer to regular until it reaches the point at which the sampling mode could be considered as regular and all DSP algorithms can be directly applicable. Unfortunately the regularization process proportionally decreases the signal sampling frequency.

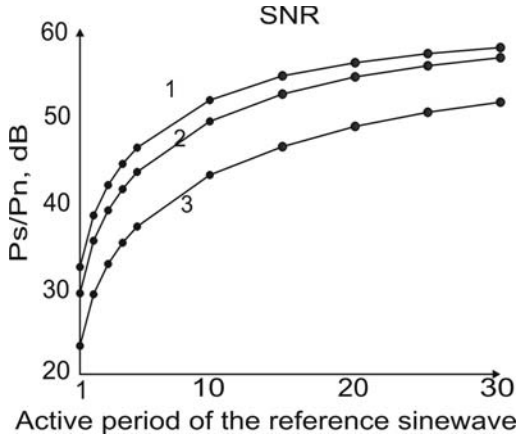


Figure 8. Signal-to-noise ratio SNR versus decimation coefficient n of the activated reference function periods for three various signals.

How it can be seen from Figure 8, the SNR is acceptable starting to 20-th activated sine-wave half period. Actually it depends on particular application.

Conclusion

Studies of SWC based DAQ lead to the following conclusions:

- Algorithms and signal processing. Signals sampled according to the SWC method has constant envelope not depending on the original analog signal frequency content. That makes it possible to develop algorithms for processing them without massive multiplication of multi-digit numbers.
- A new signal filtering algorithms, exploiting the constant envelope property, have been developed is described in [6].
- DFT without massive multiplication operation is possible.

- Applicability of traditional DSP algorithms under certain conditions has been achieved. This applicability of classical DSP algorithms can be obtained by using regularization process of SWC sampled signals.
- To gain from of the constant envelope sampling, it has to be learned how to effectively cope with the drawbacks related to the nonuniformity of the obtained digital signals. The signal regularization using the enabling function is recommended as effective tools for controlling signal sampling conditions. However it reduces the mean signal sampling rate for each DAS channel depending on particular application.
- Sampling properties. The SWC sampling properties are evaluated in the time domain and as well as in the frequency domain. Trading-off the mean sampling rate against the time resolution has to be done to provide for proper quality sampling. It is considered how the particularities of this type of signal spectra differ from spectra of the traditionally sampled signals.
- DAQ system architecture. Application of SWC sampling leads to simpler and more energy efficient hardware architectures of data acquisition systems as the signal sampling operation can be performed on the basis of a single comparator instead of ADC.

The main conclusion is that SWC signal sampling method is well suited to simultaneous multi-channel data acquisition from a large number of relatively low-frequency signal sources.

6. FPGA based implementation of the considered DAQ methods

A particular application of DASP theory for Fourier coefficient calculation has been considered. FPGA based implementation of a specific DFT multiplier-less structure (F-DFT Processor) for fast pre-processing of data has been developed and FPGA implementation specifics studied. This type of FPGA is applicable for designs of various application-specific systems fulfilling digital signal pre-processing related to data time-frequency representation, data compression/reconstruction, spectrum analysis, filtering, parameter estimation and demodulation.

Experimental system has been programmed and tested on the basis of *Altera Cyclone II EP2C70F672C6N* chip model. The core of F-DFT Processor is multiplier-less Adder-Subtractor Matrix. In addition to this Matrix, there are data input and output subsystems. Designs of these subsystems depend on specifics and requirements of various applications. Therefore the data input and output version is just one of the possible modifications. Real-time DFT and batch processing of data for DFT are two of the most often needed types of data pre-processing that can be performed on the basis of the developed FPGA chip.

Requirements dictated by a specific bioimpedance measurement system were taken into account at definition of the parameters of DFT for the specification of this FPGA chip. These parameters are the following: Fourier coefficients have to be calculated simultaneously at frequencies f0 (data mean value), f1, f2, f4, f8, f16, f32 and f64 by processing N=256 real signal sample values. The absolute values of these frequencies depend only on the parameters of data acquisition, in particular, on the specifics of the signal source and the used ADC. Thus the considered processor can perform DFT in a wide frequency range from KHz up to GHz.

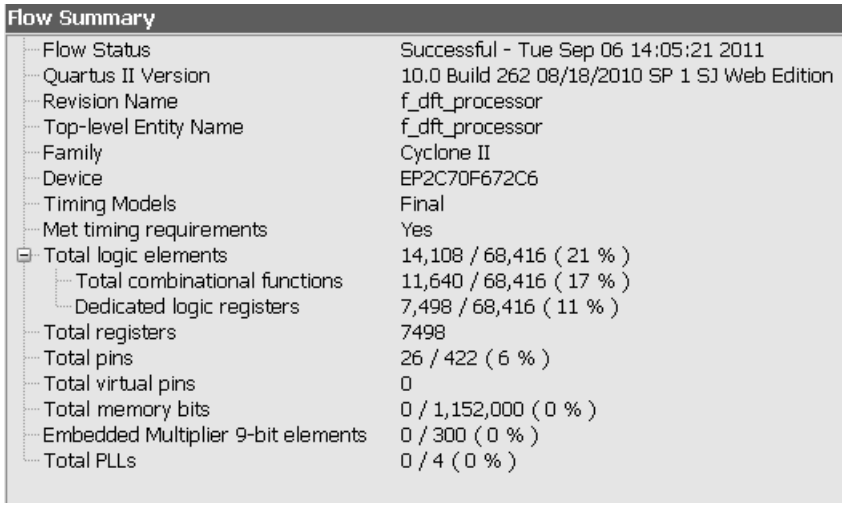
The most important parameter that has to be experimentally measured is the achieved pin-to-pin delays characterizing the Matrix, as this parameter actually defines the upper repetition rate of DFT operation or system clock frequency. Printout of so-called Clock setup, showing pin-to-pin delays for the involved data flows at the specified frequencies, follows.

Clock Setup: 'clk'

	Slack	Actual fmax (period)	From	To
1	N/A	101.49 MHz (period = 9.853 ns)	reg[231][7]	f8c7[0]
2	N/A	101.79 MHz (period = 9.824 ns)	reg[231][7]	f16c1_2[0]
3	N/A	101.82 MHz (period = 9.821 ns)	reg[109][6]	f16c3_1[0]
4	N/A	101.94 MHz (period = 9.810 ns)	reg[109][8]	f16c3_1[0]
5	N/A	102.06 MHz (period = 9.798 ns)	reg[248][7]	f32c2_1[0]
6	N/A	102.21 MHz (period = 9.784 ns)	reg[237][8]	f16c3_1[0]
7	N/A	102.26 MHz (period = 9.779 ns)	reg[136][6]	f32c2_1[0]
8	N/A	102.29 MHz (period = 9.776 ns)	reg[231][7]	f8c7[1]
9	N/A	102.45 MHz (period = 9.761 ns)	reg[147][6]	f16c3_1[0]
10	N/A	102.45 MHz (period = 9.761 ns)	reg[87][8]	f8c7[0]
11	N/A	102.53 MHz (period = 9.753 ns)	reg[231][7]	f16c1_2[1]
12	N/A	102.56 MHz (period = 9.750 ns)	reg[109][6]	f16c3_1[1]
13	N/A	102.67 MHz (period = 9.740 ns)	reg[109][7]	f16c3_1[0]
14	N/A	102.67 MHz (period = 9.740 ns)	reg[173][6]	f16c3_1[0]
15	N/A	102.68 MHz (period = 9.739 ns)	reg[109][8]	f16c3_1[1]
16	N/A	102.70 MHz (period = 9.737 ns)	reg[87][8]	f16c1_2[0]
17	N/A	102.72 MHz (period = 9.735 ns)	reg[147][8]	f16c3_1[0]

Figure 9. Printout showing test results of the F-DFT Processor measured by Quartus software. Achievable clock frequencies for various data flow routes and the related pin-to-pin delays are given for the indicated addresses.

As can be seen in Figure 9, the worst case is for the data flow in the Matrix from register reg[231][7] to the output f8c7[0]. According to Figure 9, the achievable clock frequency of the F-DFT processor under the described conditions is 101.49 MHz and the worst-case pin-to-pin delay is 9.853 ns. Required resources for the F-DFT processor design are given in the printout of Figure 10.



Flow Summary	
Flow Status	Successful - Tue Sep 06 14:05:21 2011
Quartus II Version	10.0 Build 262 08/18/2010 SP 1 SJ Web Edition
Revision Name	f_dft_processor
Top-level Entity Name	f_dft_processor
Family	Cyclone II
Device	EP2C70F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	14,108 / 68,416 (21 %)
Total combinational functions	11,640 / 68,416 (17 %)
Dedicated logic registers	7,498 / 68,416 (11 %)
Total registers	7498
Total pins	26 / 422 (6 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 10. Required resources of the designed F-DFT processor (*Altera Quartus* snapshot).

In comparison with the classical structure of the digital FIR filter on 256 coefficients design of the F-DFT processor requires much less elements, approximately 2 times less *Altera* logical elements. Performance of both devices is characterized by pin-to-pin delays and the clock frequency. To compare both, the following points have to be taken into account: (1) the considered particular F-DFT processor calculates simultaneously outputs of 15 filters (1 filter for f0 and two filters for each other 7 frequencies) in parallel; (2) the structure of the considered single FIR filter is pipelined, therefore the indicated clock frequency for it depends on the delay of only a single filter stage.

For the explained reasons, the performance of both devices cannot be directly compared. As to the resources, the requirements of F-DFT are significantly reduced.

Table 3

Summary of required resources and parameters of the F-DFT processor and a particular digital FIR filter.

Cyclone II Device EP2C70F672C6	**F-DFT data block 256**	**Pipelined FIR on 256 coefficients with 256 multipliers**
Clock frequency	101.49 MHz	117.25 MHz
Total combinational functions	11,640 / 68,416 (17 %)	28,785 / 68,416 (42 %)
Dedicated logic registers	7,498 / 68,416 (11 %)	16,807 / 68,416 (25 %)
Total logic elements	14,108 / 68,416 (21 %)	29,115 / 68,416 (43 %)
Total pins	26 / 422 (7 %)	30 / 422 (7 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)	0 / 300 (0 %)

7. Obtained research results

Overview of the specifics related to development and production of currently available DAQ systems reveals the basic tendencies characterizing the situation in this field. While about 300 industrial companies are active in this area and seemingly many various DAQ systems are offered, actually progress there is relatively slow. As it is shown the basic limiting factor is the dependency of DAQ methods on the classical theory of DSP, especially on the dominating theoretical principles of periodic sampling. Theory of randomized and alias-free signal processing is rather well suited for DAQ applications. This fact has been taken into account and the undertaken research activities of these doctoral studies are based on and exploit a number of essential results of the DASP theory. The point is that signal digitizing performed in this way is rather flexible and can be adjusted to the needs of various DAQ applications. The approach to DAQ is based on these considerations.

Summary of the obtained research results follows:

1. Research results related to exploitation of nonuniform sampling for DAQ showing that this approach has significant potential for improving DAQ. The obtained results in this area, specifically, are the following:
    - Elimination of the input signal source number dependence on the sampling rate that limits the capabilities of the classical DAQ systems. That makes possible simultaneous data acquisition in parallel from a significantly increased (up to 8-12 times) number of signal sources.
    - Development of methods and algorithms for asymmetric data compression/reconstruction (in the sense that most of the computational burden is put on the reconstruction stage), including very fast image data asymmetric compression/reconstruction.
    - Development, making and testing of an experimental complexity-reduced hardware/software system for image data compression based on this method, achieving image data compression up to 5 times for various images, including colour and standard images.
    - Development, analysis and description of various approaches to DFT coefficient estimation, in particular, for spectrum analysis of bioimpedance data obtained in result of nonuniform sampling based bioimpedance demodulation.
2. Obtained research results, related to exploitation of deterministic, randomized and pseudo-randomized quantizing methods for DAQ, are the following:
    - Results of analysis and evaluation of the impact pseudo-randomized quantizing has on the precision of spectrum analysis of data.
    - Methods for spectrum analysis of data, based on pseudo-randomized signal quantizing.

- Complexity reduced approach to bioimpedance data demodulation using DFT on the basis of rectangular functions.
- Results of comparison and evaluation of all three quantizing versions showing that pseudo-randomized quantizing usually outperforms two other quantizing options.

3. Results of research related to a new approach to DAQ based on acquiring timing information:
   - A method for DAQ, based on detection of signal and sine-wave crossing instants (SWC sampling), computer-simulated and extensively explored.
   - Computer simulation results showing the most significant advantage of SWC based DAQ systems, namely, their capability of data gathering from a very large quantity of signal sources (up to 256-512) distributed over an object of technical or biomedical origin.
   - Method for SWC sampled signal regularization that can be performed to ensure the applicability of standard DSP algorithms.
   - Special signal processing methods applicable in the case of SWC for complexity-reduced design of digital filters and multiplier-less structures for DFT [6].
4. Developed MATLAB programs for Computer simulations of the explored DAQ methods.
5. Results obtained in the experimental research area: VHDL description for FPGA implementation of a specific Fast DFT Processor, test results of this FPGA, showing, specifically, the required element count and parameters characterizing the achieved operational speed.

8. Conclusions

The goal of this work, discovering innovative methods for efficient massive data acquisition from real life objects and supplying computers with this information, has been reached and the planned tasks have been fulfilled. The developed and investigated innovative methods for complexity-reduced DAQ and energy-efficient pre-processing of the data are based on the theory, concepts and methods of the non-traditional digital signal processing DASP. The obtained research results show: (1) how that can be done and (2) what can be gained at computer system linking to the real world technical and biological objects by using the developed DAQ methods.

The basic benefits that can be obtained in this way, specifically, are the following:

1. The developed methods for multi-channel data acquisition systems are flexible and applicable for simultaneous data acquisition from many signal sources.

2. Simultaneous data acquisition from many signal sources is performed in parallel under conditions where the upper frequencies of the input signal spectra do not depend on the quantity of the input channels.
3. Input signals can be sampled directly at their sources by front-end devices that are much simpler than the traditionally used ADCs.
4. Consequently, the power consumption of these front-end devices might be significantly lower.

However DASP technology is specific. Therefore hardware and software implementing DAQ based on this technology are specific as well. MATLAB based software tools have been developed and they could be used to implement the chosen approach. Streams of digitally timed sampling events are used as digital signals representing the acquired data. Massive data acquisition is based on pseudo-randomised time-sharing, analog-to-event and time-to-digital conversions.

Experimental investigations of the developed DAQ methods were carried out by using hardware tools that have been developed in the Laboratory 2.2.of Institute of Electronics and Computer Science in the framework of ERAF Project Nr. VDP1/ERAF/CFLA/05/APK/2.5.1/000024/012 "Development of multi-channel systems for acquisitions of data from biomedical, ecological and industrial systems and transferring them to computerized systems", co-sponsored by the European Union. Author of this research work participated in this project as a member of the team.

Tests and experimental evaluation of the developed DAQ systems show that:

1. Experiments confirm the results obtained theoretically.
2. The existing microelectronic components, including FPGA chips and their programming technology, basically are suitable and can be used for implementation of gathered data specific pre-processing.
3. Potential of FPGA usage for specific data pre-processing is demonstrated by performance of the developed Fast DFT Processor (see Protocol of F-DFT Processor international evaluation results given in Attachment).
4. Attempts to use the existing microelectronic components for implementation of the front-end operations at DAQ show that this approach basically does not lead to sufficiently high results.
5. To fully gain from the research results obtained in the framework of this work, implementation of the front-end DAQ procedures, defined by the developed DAQ methods, should be based on developed new specific ASICs (Application Specific Integrated Circuits).

References

1. Bilinskis I., Sudars K. *Processing of signals sampled at sine-wave crossing instants.* Proceedings of the "2007 Workshop on Digital Alias-free Signal Processing" (WDASP'07), London, UK, 17 April 2007, pp. 45-50.
2. Sudars K., Ziemelis Z., *Expected performance of the sine-wave crossing data acquisition systems,* Proceedings of DASP Workshop, London, UK, 17 April 2007.

3. Bilinskis I., Sudars K. *Digital representation of analog signals by timed sequences of events*, "Electronics and Electrical Engineering", No. 3(83), March, 2008, Presented in International Conference "Electronics 2008", Kauna, Lithuania, 20-22 May 2008.
4. Bilinskis I., Sudars K. *Specifics of constant envelope digital signals*, "Electronics and Electrical Engineering", No. 4(84), April, 2008, Presented in International Conference "Electronics 2008", Kauna, Lithuania, 20-22 May 2008.
5. Artyukh Yu., Bilinskis I., Sudars K., Vedin V., *Multi-channel data acquisition from sensor systems.* Proceedings of the 10th International Conference "Digital Signal Processing and its Applications" (DSPA'2008), Moscow, Russia, 2008, Vol.X-1, pp.117-119.
6. Artyukh Y., Bilinskis I., Sudars K., European Patent Application No. EP2075912 A1, *Method for complexity-reduced digital filtering and parameter estimation of analog signals*, Assignee: Institute of Electronics and Computer Science of Latvia, European patent Bulletin, January 7, 2009.
7. Bilinskis I., Sudars K., Min M., Annus P., *Advantages and limitations of an approach to bio-impedance data acquisition and processing relying on fast low bit rate ADCs*, Baltic Electronic Conference BEC 2010, Tallinn, Estonia, 4.-6. October 2010.
8. Artyukh Y., Bilinskis I., Roga S., Sudars K., *Digital Representation of Analog Signals leading to their Energy-efficient Processing*, Green Information Technology (GREEN IT 2010), Singapore, 25.-26. October 2010.
9. Sudars K., *Data Acquisition Based on Nonuniform Sampling: Achievable Advantages and Involved Problems*, "Automatic Control and Computer Science" Magazine, Allerton Press, 2010, Vol. 44, No. 4, pp. 199-207.
10. Bilinskis I., Skageris A., Sudars K. *Method for fast and complexity-reduced asymmetric image compression*, "Electronics and Electrical Engineering", No. 4(110), May, 2011, Presented in International Conference "Electronics 2011", Kauna, Lithuania, 17-19 May 2011.
11. Artyukh Y., Bilinskis I., Rybakov A., Sudars K., Vedin V., Modular Multi-channel Data Acquisition Systems, "Automatic Control and Computer Sciences" Magazine, Allerton Press, 2008, Vol. 42, No. 3, pp. 113–119.
12. Artyukh Y., Bilinskis I., Boole E., Rybakov A., Vedin V., "Wideband RF signal digititizing for high purity spectral analysis," Proceedings of the 2005 International Workshop on Spectral Methods and Multirate Signal Processing, (SMMSP 2005) June 20-22, 2005, Riga, Latvia. pp. 123-128.
13. Artyukh Y., Bilinskis I., Greitans M., Mednieks I., Rybakov A., DASP-Lab System – a demonstrator of the new DSP technology. //Automatic Control and Computer Science, Allerton Press, 2000, No. 6, pp. 3-21.
14. Artyukh Y., Bilinskis I., Rybakov A., Stepin V., "Pseudo-randomization of multiplexer-based data acquisition from multiple signal sources," Proceeding of DASP Workshop, London, UK, 17 April 2007.
15. Bilinskis I., *Digital Alias-free Signal Processing*, Wiley, 2007, ISBN: 978-0-470-02738-7
16. Bilinskis I., Mikelsons A., *Randomized Signal Processing*, Prentice Hall, 1992, ISBN: 0-13-751074-8
17. Bilinskis I., Rybakov A., "Iterative spectrum analysis of nonuniformly undersampled wideband signals," Electronics and Electrical Engineering, 2006, vol. 4(68), pp. 5-8.
18. Blackledge J. M., Burge R. E., Barratt N. R., "Phase imaging by real-zero conversion and its applications to synthetic aperture radar", Journal of Physics D: Applied Physics, Volume 20, Issue 11, November 14, 1987
19. Borokhovych Y., Gustat H., "4-bit, 15 GS/s ADC in SiGe," NORCHIP 2008, 16-17 Nov. 2008, Tallinn, Estonia, pp. 268-271.
20. Candès E. J., Wakin M. B., "An Introduction to Compressive Sampling", IEEE Signals Processing Magazine, March 2008
21. Contadini F., "Oversampling with averaging to increase ADC resolution", Internet: http://www.embedded.com/design/223000385, 1 March 2010 15

22. Daria V. R., Saloma C., "High-accuracy Fourier transform interferometry, without oversampling, with a 1-bit analog-to-digital converter", Applied Optics, Volume 39, Issue 1, January 1, 2000, pp. 108-113
23. Eng F., *Non-Uniform Sampling in Statistical Signal Processing*, Dissertation at Linköpings Universitet, Sweden, 2007, ISBN: 978-91-85715-49-7
24. Internet: http://en.wikipedia.org/wiki/Data_acquisition
25. Internet:http://www.datatranslation.com/docs/whitepapers/benefits_simulataneous_daq.pdf
26. Internet: http://www.radiolocman.com/news/new.html?di=64921
27. Koch P., Prasad R., "The Universal Handset," IEEE Spectrum, vol. 46, pp. 32-37, April 2009.
28. Kumaresan R., Wang Y., "On representing signals using only timing information", The Journal of the Acoustical Society of America, Volume 110, Issue 5, November 2001, pp. 2421-2439
29. Lim M., Saloma C., "Direct signal recovery from threshold crossings", Physical Review E (Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics), Volume 58, Issue 5, November 1998, pp. 6759-6765
30. Logan Jr. B. F., "Information in the zero crossings of bandpass signals", Bell System Technical Journal, Volume 56, April 1977, pp. 487-510
31. Marvasti F. (editor), *Nonuniform Sampling Theory and Practice*, Kluwer Academic/Plenum Publishers, 2001, ISBN: 0-306-46445-4
32. Marvasti F., *A Unified Approach to Zero-crossings and Nonuniform Samplingof single and multidimensional signals and systems*, Illinois Institute of Technology, 1987, ISBN: 0-9618167-0-8
33. Min M., Bilinskis I., Artyukh Y., Annus P., Land R., Märtens O., "Nonuniform Sampling and Demodulation for Wideband Multi-frequency Bioimpedance Measurements," Proceedings of the Workshop on Digital Alias-free Signal Processing, London, 17 April 2007
34. Nait-Ali A., Cavaro-Menard C. (editors), *Compression of Biomedical Images and Signals,* Wiley, 2008, ISBN: 978-1-84821-028-8
35. Nazario M. A., Saloma C., "Signal recovery in sinusoid-crossing sampling by use of the minimum-negativity constrain", Applied Optics, Volume 37, Issue 14, May 10, 1998
36. Papenfus F., *Digital Signal Processing of Nonuniformly Sampled Signals*, Dissertation at Universität Rostock, Publisher: Shaker Verlag, 2008, ISBN: 978-3-8322-6895-4
37. Raghavendra C. S., Sivalingam K. M., Znati T. (editors), *Wireless Sensor Networks*, Kluwer Academic, 2004, ISBN-13: 978-1402078835
38. Rieke F., Warland D., R. R. van Steveninck, Bialek W., "Spikes: Exploring the Neural Code", MIT Press, Cambridge, MA, 1997
39. Sundstrom T., Alvandpour A., "A 2.5-GS/s 30-mW 4-bit Flash ADC in 90nm CMOS," NORCHIP 2008, 16-17 Nov. 2008, Tallinn, Estonia, pp. 264-267.
40. Zeevi Y. Y., Gavriely A., Shamai (Shitz) S., "Image representing by zero and sine-wave crossings", Journal of the Optical Society of America A, pp. 2045-2060, 1987

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2012. Vol. 787
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 79–105 P.

Design and Implementation of MansOS: a Wireless Sensor Network Operating System

Atis Elsts[1,2], Girts Strazdins[1,2], Andrey Vihrov[1,2], and Leo Selavo[1,2]

[1] Faculty of Computing, University of Latvia,
19 Raina Blvd., Riga, LV-1586, Latvia
[2] Institute of Electronics and Computer Science,
14 Dzerbenes Str., Riga, LV-1006, Latvia
atis.elsts@lu.lv, girts.strazdins@lu.lv,
andrejs.vihrovs@edi.lv, leo.selavo@lu.lv

Abstract. Wireless sensor networks (WSN) consist of a number of low-power spatially distributed autonomous nodes equipped with means of communication. Developing software for WSN is challenging; operating system (OS) support is required. In this paper we present MansOS, a portable and easy-to-use WSN operating system that has a smooth learning curve for users with C and UNIX programming experience. The OS features a configuration model that allows to reduce application binary code size and build time. In contrast to other WSN OS, MansOS by default provides both event-based and threaded user application support, including a complete although lightweight implementation of preemptive threads.

Keywords: Sensor networks, operating systems.

1 Introduction

Few areas of embedded system programming require OS support more than WSN programming does. Networks formed by autonomous small-scale devices with tight resource and energy constraints are systems of great complexity. Developing fully application-specific solutions without using middleware or OS is not an option for majority of users. MansOS is an open-source operating system designed to serve their needs.

WSN programming is challenging because it brings together complexity of embedded device programming and complexity of networked device programming. Therefore an easy-to-use OS with a smooth learning curve is needed. Since a lot of system programmers have experience with C programming and UNIX-like concepts, it makes sense to adapt these concepts to WSN programming. MansOS is written in plain C, aims to be user-friendly and use familiar concepts.

In contrast to their desktop counterparts, embedded hardware architectures and platforms come in great variety and are often specially adapted to concrete applications. Therefore portability is a critical requirement for embedded software. MansOS is portable and runs on several WSN mote platforms.

The design and implementation of a feature complete WSN OS is not a quick or easy task. In some form, MansOS has been under development since 2007. It started off as a LiteOS clone: an attempt to bring portability to this WSN OS. During the development, MansOS has been influenced by ideas from Contiki and Mantis.

Existing WSN OS in some cases use unnecessarily heavy technologies and suboptimal implementations. For example, OS support for threads can be simplified and optimized by taking into account traits specific to WSN OS software: the low number of total threads expected and cooperativeness of user threads. Moreover, code of existing WSN OS is often bloated by forcing the use of unnecessary resources and components. MansOS brings these simplifications and optimizations to WSN OS area, and allows smaller resource use granularity.

The paper starts with an overview of related work. Then a section is devoted to description of MansOS architecture, and hardware design space. It then proceeds with a description and evaluation of selected components, and concludes with a comparison of MansOS and competing solutions.

2 Related work

Multiple operating systems for sensor networks have been proposed previously, focusing on different design aspects. Notable examples include TinyOS, Contiki, LiteOS and Mantis.

2.1 TinyOS

TinyOS [11] is an actively supported and well tested WSN OS, has created a wide contributor and user community, and can be considered *de facto* standard for WSN programming. It is primarily targeted to sensor network researchers.

Compact, reactive scheduler is used (core system uses 400 bytes of program memory). Source code is written in *nesC* language, a C dialect processed by a *nesC* parser and pre-compiled into a single C source file. This single file is then compiled into a firmware image and takes advantages of static compiler optimizations.

TinyOS is a highly modular system, consisting of components, wired together using specified interfaces. Each component provides a particular service and interfaces describe *commands* for starting a service and *events* for signalling completion of a service routine. Inside components low-priority tasks are scheduled, using non-preemptive, run-to-completion FIFO task queue. High-priority event handlers are used for time-critical section execution. Optional preemptive scheduler can be used, implemented as an add-on, called TOSThreads [10].

Although TinyOS is portable, it has a steep learning curve for novice WSN developers. The main reasons of TinyOS complexity are:

- event-driven nature of TinyOS. Event-based code flow is more complex for programmers to design and understand, as state machine for split-phase operation of the application has to be kept in mind;

- nesC language concepts, unfamiliar even to experienced C and embedded system programmers.

These limitations are in the system design level, and there is no quick fix available. The most convenient alternative is to implement middleware on top of TinyOS for simplified access to non-expert WSN programmers.

2.2 Contiki

Contiki is a lightweight operating system with support for dynamic loading and replacement of individual programs and services. It is built around an event-driven kernel and provides optional preemptive multithreading [6]. Contiki is written in C language and has been ported to a number of platforms, including TelosB and Zolertia Z1, having different CPU architectures: Atmel AVR, Texas Instruments MSP430 and others.

The only abstractions provided by Contiki kernel are CPU multiplexing and dynamic program and service loading. Additional abstractions are provided by libraries with full access to underlying hardware. Loadable programs are implemented, using modified binary format containing relocation information and performing run-time relocation.

Contiki provides proto-threads abstraction: ability to write thread-like programs with blocking calls on top of event-driven kernel [7]. Each proto-thread requires only two bytes of memory, and has no separate stack. As an alternative to cooperative proto-threads, Contiki provides preemptive threading model implemented as optional library. It uses separate stacks for each thread and context switching, which must be implemented for every CPU, if used. However, to the best of our knownledge, there are no platforms whose implementations of threads support preemption.

Process is either an application or a service – a process implementing functionality used by multiple applications. Both applications and services may be replaced at run-time. Communication between services is implemented using event passing through kernel.

The porting of Contiki consists of writing platform-specific boot up code, device drivers, architecture specific dynamic program loading and context switching for preemptive multithreading. The kernel and service layer are platform-independent.

Contiki programs are relatively heavy-weight, as they usually use several kilobytes of RAM and have program code more than 10 kilobytes in size. Core routines of the system are sometimes duplicated between platforms, for example, timer interrupt handlers are custom for every MCU architecture. Architecture-independent scheduler of preemptive threads is a desired feature that is not included in the OS by default.

2.3 LiteOS

LiteOS is a multithreaded operating system that provides Unix-like abstractions for wireless sensor networks [4]. It offers hierarchical file system, remote shell,

dynamic application loading, preemptive scheduler for multithreaded applications and and object oriented programming language LiteC++ – a subset of C++. LiteOS has been implemented on MicaZ and Iris mote platforms, both with AVR microcontrollers.

LiteOS utilizes a specific binary image format containing relative addresses to provide dynamic application loading.

LiteOS includes a hierarchical, UNIX-like file system, called LiteFS, for storage of application sensor data and application binary images. All files, including device drivers, provide unified POSIX-compatible access functions: `fopen()`, `fclose()`, `fseek()`, `fread()`, `fwrite()`, etc. Therefore also all sensor node devices are included in the file system accessible from remote shell.

Split phase operations, such as sending a radio message and waiting for end of transmission, are implemented as blocking calls. External events, such as reception of a radio message or external pin interrupt, were implemented as callback function handlers in the first versions of LiteOS, but were transformed to blocking calls later. Therefore LiteOS provides fully threaded programming with blocking calls, and no event callback handling.

To access kernel functions from user threads, system calls or *call gates* are used. Therefore function implementations can be changed at run-time, and kernel image updated without modification of user application.

The source code is 8-bit AVR platform-specific and significant changes are required to port LiteOS to other platforms with other microcontrollers.

2.4 Mantis

Mantis is a multithreaded cross-platform embedded operating system for wireless sensor networks, supporting complex tasks such as compression, aggregation and signal processing, implemented in a lightweight RAM footprint that fits in less than 500 bytes of memory, including kernel, scheduler, and network stack [3]. Mantis is implemented on multiple platforms, including PCs and PDAs, allowing to create hybrid networks consisting of real sensor nodes and virtual ones, simulated on top of one or multiple PCs. Written in C language, Mantis OS translates to a separate API on the PC platform, which can be augmented by any other required functionality – graphical user interface, data bases, web services.

Mantis provides preemptive, time-sliced thread scheduling and synchronization using semaphores. Each thread occupies a separate stack. Each context switch is performed by the kernel task scheduler in timer interrupt handler, and uses only approximately 120 instructions. All other interrupts besides timer are handled by device drivers directly.

Additional abstraction layer is implemented in the kernel, providing blocking call interface for external event waiting (radio or UART packet reception). Data buffer pool is used to share buffers between all communication layer services.

Mantis kernel code is platform-independent. However, platform- and chip-level code is mixed, there are no TelosB or MicaZ platforms, only MSP430 and AVR code, which is microcontroller (MCU) or architecture specific. Separation of MCU architectures, specific chips and platforms would improve portability.

3 MansOS architecture

In the first part of this section the MansOS hardware abstraction model is specified. After that, the design space of a portable WSN OS is outlined, by describing the specific architectures and hardware modules we chose to support.

3.1 Abstraction model

As one of our design goals is minimization of effort required to port the OS to new WSN hardware platforms, MansOS provides modular architecture. Chip-specific code is separated from platform-specific and platform-independent routines. Driver code is designed to be platform-independent where possible, therefore a single MansOS driver frequently is usable across multiple platforms.

Hardware abstraction model in MansOS (Fig. 1) is based on a key observation from [9]: due to requirements of energy efficiency in WSN it is not enough to expose only a single, strictly platform-independent hardware abstraction layer. The users should be allowed to exploit device-specific hardware features for increased efficiency and flexibility.

In MansOS the user has access to all four hardware abstraction layers:

- device-specific code (placed in directory `chips`) – drivers for individual devices and microcontrollers;
- architecture-specific code (directory `arch`) – code particular to a specific architecture (such as MSP430 or AVR);
- platform-specific code (directory `platforms`) – code particular to a specific platform (such as Arduino, TelosB or Zolertia Z1).
- platform independent code, including the *hardware interface layer (HIL)*, directory `hil`.

The HIL code provides unified device interface for kernel and user applications. Wiring, function binding and platform or architecture-specific constants are defined at `arch` and `platform` levels. To take an example, radio driver's interface is defined in the HIL level. During compilation time, the interface is bound to a specific implementation, which is chosen at the `platform` level, containing the glue code. For TelosB platform, CC2420 radio driver is chosen, and so on.

The model explicated here is similar to the one found in Contiki: `platforms` directory in MansOS roughly corresponds to `platforms` directory, `arch` to `cpu` in Contiki, `chips` to `core/dev`, and the rest of MansOS system (`kernel`, `hil`, and `lib`) to the rest of `core` folder in Contiki. The chief difference between these systems is better organization of chip- and platform-specific code in MansOS; for example, the periodic timer interrupt handler code (the "heartbeat" of the system) is unified and shared by all platforms. Another difference is function binding: in MansOS it is done earlier, at compile time. This design decision allows reducing both binary code size and RAM usage, as well as run-time overhead. To take a concrete example, in Contiki the radio driver is accessed through function

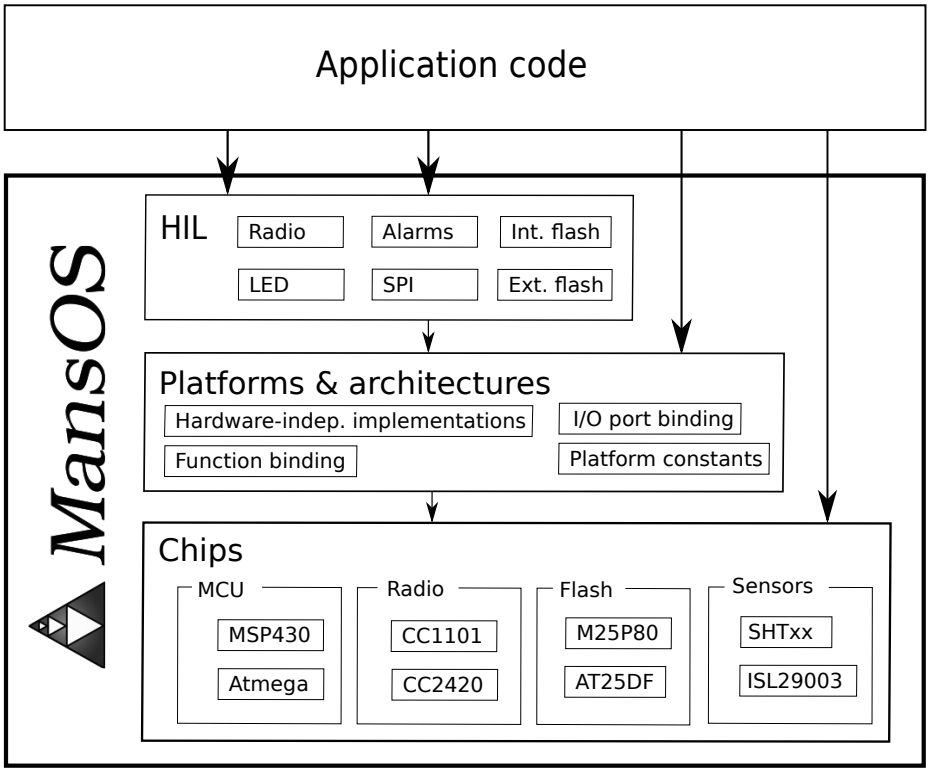


Fig. 1: MansOS components and abstraction layers

pointers in `struct radio_driver` structure. The structure itself takes twenty bytes in RAM. Furthermore, indirect function calls have to be used, which adds two byte flash usage overhead for each call, as well as CPU run-time overhead, because an extra `mov` instruction is generated. The extra `mov` takes two CPU cycles to execute, because on MSP430 instruction execution takes an extra CPU cycle for each memory access. In MansOS all calls are direct (glued by inline functions or macros), therefore extra resources are not used.

Similar parallels can be drawn between MansOS and TinyOS, although the latter lacks explicit separation of architecture-specific code: `platforms` in MansOS maps to `platforms` in TinyOS, `chips` to `chips`, `hil` to `interfaces`, `kernel` to `system`, `lib` to `lib`. A notable difference is the impossibility of direct hardware component access in TinyOS application code. It could be argued that this restriction leads to a better code organization, but we feel that it is too limiting to the user.

As the analysis shows (Table 1), approximately half of total MansOS code is hardware-independent. Since the amount of hardware-dependent code varies greatly with the number of hardware platforms supported, comparison is more fair when a specific platform is fixed. When TelosB is selected as the platform,

Table 1: Source code size breakdown with regard to MansOS components, lines of code (excluding comments and empty lines)

	All platforms		**TelosB**	
Chip-specific code	7132	34.61%	2657	19.86%
Architecture-specific code	2063	10.01%	582	4.35%
Platform-specific code	1482	7.19%	208	1.56%
Interface layer code	1814	8.8%	1814	13.56%
Kernel code	1240	6.02%	1240	9.27%
Network protocol code	3683	17.87%	3683	27.53%
File system code	1384	6.72%	1384	10.35%
Library code	1809	8.78%	1809	13.52%
Total device-independent code	9930	48.19%	9930	74.23%
Total code	**20607**	**100%**	**13377**	**100%**

only a quarter of the code turns out to be chip- or platform-specific. Most of the hardware-dependent code is plain C; ASM is used only in a few, specific places, such as thread context switching.

3.2 Hardware support

A wireless sensor network consists of a number of distributed devices that are not only small and low-cost, but also have to be powered from relatively low capacity batteries. Therefore computing power limitations are extremely tight. Typical WSN mote resources include ten to a few hundreds kilobytes of program memory, a few kilobytes of RAM, no memory protection or mapping support. The microcontroller usually has a few MIPS of computing power and features several lower energy consumption modes. A WSN OS has to make good use of the limited resources available. Energy efficiency is of paramount importance: the OS has to provide options for low duty cycling.

Among typical architectures used in for WSN sensor motes and resource constrained embedded devices, Texas Instruments MSP430 and Atmel AVR are prominent.

MSP430 [14] is a 16-bit, low-power microcontroller using MIPS architecture. The device features a number of peripherals useful for a WSN device. Digital and analog I/O pins are provided, as are multiple hardware timers, including pulse-width modulation (PWM), and watchdog. Analog inputs can be sampled using the built-in ADC circuitry, while digital pins allow specific data transfer protocols to be used, for example, U(S)ART, SPI, and I^2C access. Notable platform examples are TelosB-compatible motes, such as Tmote Sky [13], as well as newer developments like Zolertia Z1 [16].

Atmel AVR [2] is an 8-bit modified Harvard architecture RISC microcontroller. Integrated ADC, watchdog and multiple timers with PWM are present as well, as are digital and analog I/O ports. One difference between the two

architectures is related to flash memory access: MSP430 use unified memory address space for RAM and flash, while in AVR macros have to be used for program memory access. Equivalently powerful AVR chips use more energy than MSP430, therefore they are better suited to application domains where energy requirements are less stringent, such as automotive applications or building automation. Notable platform examples include Arduino [1] and Waspmote [12], both using *Atmega* series MCU.

Several peripherals (sensors and actuators) are usually present on the mote. The OS therefore has to provide support for digital data transfer bus protocols (UART, SPI, I^2C) typically used for communication with the peripherals. At least, API to the MCU built-in hardware support has to be provided. However, hardware-only support is not sufficient for all times, as our experience shows. For example, several slightly different I^2C protocol versions are used on different peripheral devices (such as light sensors). The hardware support for I^2C on MSP430 fails to take into account these differences. Therefore, a configurable software implementation of the protocol has to be provided by the WSN OS in order to properly communicate with these devices. Finally, platform-independent API for the most popular sensors (voltage, light, humidity and temperature) can be expected.

Time accounting is an essential feature of the WSN OS, since WSN users often require sensor measurements to be timestamped. As real time clock (RTC) chips are seldom present on WSN motes, the OS has to emulate one using MCU hardware timers.

Finally, support for at least wireless communication has to be included in the OS, since it is by far the most popular form of communication used in sensor networks. A frequently encountered design option is IEEE 802.15.4 compatible transceiver chips using 2.4 GHz frequency band. Support of such a chip can be expected from the WSN OS. Support for IEEE 802.15.4 MAC layer is optional, as WSN applications typically use WSN-specific MAC protocols.

4 MansOS components and features

This section describes selected MansOS components in detail, namely the configuration mechanism, kernel, threads, file system, and the reprogramming mechanism. The section is concluded with a technical discussion about usability and portability. Although interesting, the description of MansOS network stack goes beyond the scope of this paper. This custom stack has support for network addressing, MAC protocols, multi-hop routing, and pseudo-sockets. IPv6 support is available as an external third-party library by using uIPv6 [8].

4.1 Configuration mechanism

Many users are worried that using a WSN OS as opposed to writing all code in application-specific way leads to bloated code sizes and inefficient resource usage. MansOS configuration mechanism is designed to deal with these problems. As

non-intrusiveness is one of our design goals, MansOS provides reduced program sizes and seamless OS integration with application code. In fact, no MansOS services are required to be used – the OS can function as a simple library of frequently used routines.

MansOS configuration mechanism is a key feature of the system, underlying whole component selection and implementation. The mechanism is based on observation that the need for run-time re-configuration or WSN applications is small. In contrast to desktop systems, where threads and processes are created and die constantly, on small resource-constrained systems resource allocation is usually static. Consequently, the allocation can be done at compile time.

The benefits of rich compile-time configuration include:

- code size reduction by explicitly selecting used and unused components;
- more flexible resource usage by providing custom, more compact versions of the code for most frequenctly used scenarios, and for cases when resources are severely constrained. For example, a simplified scheduler is used in the default case of only two threads;
- application code complexity reduction, because run-time reconfiguration support in applications becomes less important. Run-time adaptation to resource allocation is often not necessary, since the compile-time system is flexible enough;
- run-time overhead reduction by compile-time binding. This allows both reducing processing overhead, since direct function calls are cheaper than calls by pointer, and reducing RAM & flash usage overhead, since there is no need to store device driver structures for indirect access.

The objective of the configuration mechanism is to achieve the modularity and heavy optimizations made possible by using *nesC* in TinyOS, but without the complexity of having to learn a new programming language. Therefore, the challenge is to emulate specific features of component-oriented programming using plain C and GNU make.

The interactive part of the configuration mechanism is implemented using configuration files. The files are hierachical: a system-wide default configuration template is used as the base, to which platform-specific, site-local, and application-specific changes are added. Relations between components are possible: there can be either a dependence relation (A requires B) or conflict relation (A cannot be used together with B).

The non-interactive part is implemented using GCC and GNU binutils support. The optimization has two independent stages. First, after the compilation process all object files are sorted in two sets: the set reachable (via function calls) from user code and the set unreachable from user code. Only the reachable files are passed to the linker. The second stage is based on a linker feature which allows to discard unused code sections. The method can be used only when each function has been put in separate code section by the compiler, but provides finer-grained optimization if active.

4.2 Kernel

In an embedded operating system, the two main functions of OS kernel are the initialization of the system and execution of the main loop. The kernel should be as small and non-intrusive as possible and feature energy saving support.

Initialization MansOS components are initialized in the `main()` function. First, for platform-specific initialization, `initPlatform()` routine is called. This routine is custom for each platform. Second, generic component initialization is done, as the latter can depend on the former. The next action taken after all initialization is completed, depends on programming model used. For event-based execution, `appMain()` is called. For threaded execution, two threads (user and kernel) are created and OS scheduler started.

Alternatively, by specifying a configuration option, the user can completely disable kernel code. The only requirement in that case: all components used should be properly initialized from user code.

Execution models Two application execution models are used in WSN OS:

- event-based (asynchronous);
- thread-based (synchronous).

Event-based model is simpler and requires less resources: scheduler code is not included in the OS, and thread stacks do not use extra RAM. On the other hand, this model is more challenging for the programmer, especially for one who is developing lenghty applications. For event based execution, program flow is not reflected in the source code. In this way event-based programming is similar to using `goto` operator, as in both cases the user has to keep in mind a complicated mental model of program's states.

The benefits of thread-based model can be observed in application code, as it becomes easier to write and understand. On the other hand, this approach is not only more heavyweight, but application execution becomes more difficult to trace, stack overflow errors as well as race conditions become possible, and the OS kernel becomes more challenging to implement correctly. Taking all this into account, MansOS offers both models and lets the user choose.

Event-based execution This is the default implementation used in MansOS. In event-based execution model, the user registers *callbacks* and writes code for callback handler functions.

Take software timers (named *alarms* in MansOS) as an example. Alarm callback function pointers are put in a global list, ordered by alarm firing time. The list is processed in the periodic timer interrupt handler, executed 100 times per second (user-configurable value). Therefore, timers with precision up to 10 ms are available by default.

Similar callbacks can be registered for packet reception, whether serial or radio. User callbacks are executed immediately after hardware signals arrival of

new data, therefore delay is the smallest possible. However, user callback code is executed in the interrupt context and can cause problems: either if the execution blocks for too long, or if the user code re-enables interrupts. In the first case, the result is a completely blocked system. In the second case, nested interrupts become possible, so all of OS code has to be reentrant.

Energy efficiency in this model can be achieved by calling one of `sleep()` family functions in application's main loop.

Threaded execution Thread implementation in a WSN OS can be simplified if two observations are taken into account. First, the number of threads typically required by a WSN application is small. In most of cases, as single user thread is sufficient, if blocking function calls are allowed in it. Second, in contrast to desktop OS, threads in WSN OS can be expected to be cooperative. The first observations motivates the OS to provide simpler scheduler version by default, supporting only two threads. The second allows to forget about time-slicing and similar fairness guarantees.

Correct locking is a big issue in multithreaded software architectures. If the locking is not correct, race-conditions can lead to corrupt data, or deadlocks can occur. Even if the locking is correct, significant code size overhead still remains. The locking in a WSN OS kernel can be simplified by making the kernel thread to run with higher priority. MansOS thread implementation is hierarchical: user threads are one hierarchy level below the kernel thread. The kernel thread is used for system event processing only and cannot be interrupted by user threads, while user threads can interrupt each another.

At least two threads are always created: a user thread and the kernel thread. Multiple user threads are optionally available. In the latter case, two scheduling policies are available: *round-robin*, in which the least recently run user thread is always selected, and *priority-based*, in which the thread with the highest priority is always selected (from all threads that are ready to run).

Mutexes are available as means of synchronization. Sequential execution of two threads can be implemented using a mutex.

Listing 1 Thread stack guard

```
#define STACK_GUARD() do {                                          \
        /* declare a stack pointer variable */                      \
        MemoryAddress_t currentSp;                                  \
        /* read the current stack pointer into the variable */      \
        GET_SP(currentSp);                                          \
        /* compare the current stack pointer with stack bottom, */ \
        /* and abort in case of overflow */                         \
        ASSERT_NOSTACK(currentSp >= STACK_BOTTOM());                \
    } while (0)
```

Stack overflow is a nasty and hard-to-detect problem when threads with small and constant-sized stacks are used. To alleviate the detection of this problem, MansOS includes *stack guards* (Listing 1) – code fragments that can be put in functions most likely to be in the bottom of the call chain. The guard immediately aborts program execution in case an overflow is detected.

Energy efficiency using threaded execution can be achieved by calling one of `sleep()` functions in the main loops of every user thread. The system will enter low power mode if no threads (including the kernel thread) are active.

4.3 File system

A typical task for a WSN node is data logging for later relaying and analysis, since immediate transmission is not possible in all cases. Most WSN nodes include a flash chip for this purpose. However, using these chips directly by low-level device commands is non-trivial. Often it is needed to distinguish amongst several logical data streams and dynamically allocate space between them, as well as deal with the chips' hardware limitations. A WSN operating system should therefore provide a clean and easy interface to the data storage and deal with the hardware details.

MansOS features a simple file system that abstracts the physical storage as a number of logical files or streams. Following the MansOS philosophy, the file system interface is synchronous (UNIX-like) and thread-safe. In addition to basic file commands, the system has non-buffering and integrity-checking modes. On the low level, the system is designed for flash chips that have very large segments and don't contain integrated controllers that handle data rewrites and wear levelling.

A flash memory segment is the minimal unit of memory cells that can be erased at once (flash memory cells need to be erased before repeated writes). Segments can be several hundred kilobytes big depending on the flash type and model.

Data organization The file system divides physical storage – flash memory – in *data blocks* of fixed size. A file is a linked list of data blocks; new blocks are allocated on demand. Contrary to the contiguous storage approach used by some WSN file systems (Coffee [15]), this allows for dynamic file sizes at no cost. The next block's number is stored at the end of the current one.

The size of a data block is chosen so that there is low overhead from traversing and allocating blocks, yet so that there isn't much space loss from incomplete blocks. One flash segment contains a small number of data blocks, so that there is smaller chance for multiple files to occupy one segment. On the TelosB platform, which has a flash with 64 KB big segments, they are divided in four 16 KB data blocks, giving the total of 64 data blocks chip-wide.

For integrity checking, data blocks are further divided into *data chunks*, which fit into the WSN node's memory and have a checksum appended. This allows to detect errors without reading the data twice. The overall division of flash memory into smaller elements is shown in Fig. 2.

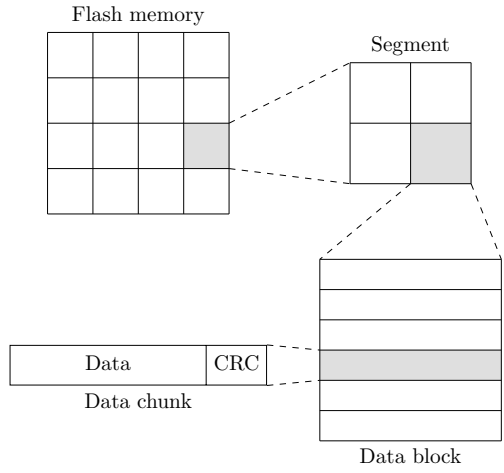


Fig. 2: Structural elements in the flash memory

Flash memory limitations on rewriting individual cells make the naïve approach of updating a file's contents in-place impractically slow. Some implementations use log-structured file system approach to solve this (the ELF file system [5]). But, since sensor data are sequential, the benefit of data rewrites may not justify the complexities they incur. Following the "keep it simple" principle, the MansOS file system disallows data rewrites completely; data can only be appended to a file.

Data block management Information about data blocks is held in the *block table*, a bitmap containing the current state of each data block (Fig. 3). The block table is small enough to be stored in the WSN node's EEPROM memory, where it can also be easily updated.

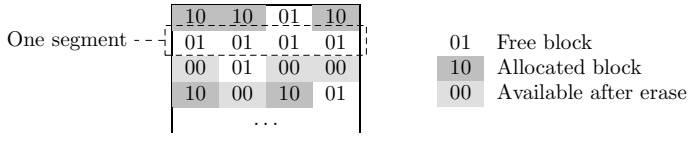


Fig. 3: Block table format

A data block can be in one of the three states: 1. free; 2. allocated; 3. available after erase (the erase operation needs to be performed on the block before it is usable again). After first time initialization, all blocks are in state 3.

The data block allocation procedure searches for usable blocks in the block table and assigns them to files on demand. It also attempts to decrease the number of blocks in state 3 that share a segment with another file (and cannot

be readily used) and to equalize the number of erases each data block is subjected to, thus performing flash memory wear levelling.

For this, free data blocks are probed in the following sequence, until one is found:

1. free blocks in the same segment as the previous data block of the file in question;
2. blocks in an empty segment;
3. blocks in an empty segment that needs to be erased before use;
4. other free blocks.

At each step, the block to allocate is chosen randomly from all available.

In the worst case, for steps 2–4 the procedure has to look at all data blocks in all segments. This can be improved by bit-packing data block states in one segment into one machine word and using bitmasks to determine the overall state of each segment.

Control structures File entries are kept in the root directory, which is also stored in the EEPROM memory. A file entry contains file name, first block number, file size and other fields. To keep the code size smaller, there is no support for hierarchical directory structure.

In-memory, open files are represented by two-tier structures, where a common part contains a file entry cache, reference count and a synchronization mutex, while the per-thread parts store file positions and read/write buffers. The use of buffers allows the flash chip to be in low-power mode most of the time.

4.4 Run-time management and reprogramming

Management Occasionally WSN applications require interactive management of specific resources, whether sensors, actuators, or software variables. Management interface is also a useful tool to non-programmers; it gives them a hands-on experience with the network.

In order to perform run-time management an efficient protocol (named *SSMP*) is implemented. Each available resource on the mote is assigned an object ID, which uniquely identifies the resource in mote-local scope. The object IDs form a hierarchical space; for example, object ID for LEDs is a prefix for object ID for red LED. In this way, multiple resources can be accessed at once, by specifying only the common prefix of their object IDs.

A command line shell is implemented to provide user access to the management interface (Fig. 4). The shell can be used both interactively and non-interactively from scripts. The shell allows to access arbitrary object IDs, although shortcuts for the most frequently used are present, for example `led` (LED control and status) and `sense` (sensor status). The shell can function in broadcast or unicast modes; in the first, all reachable motes in the network are accessed; in the second, only a single mote specified by its address is queried.

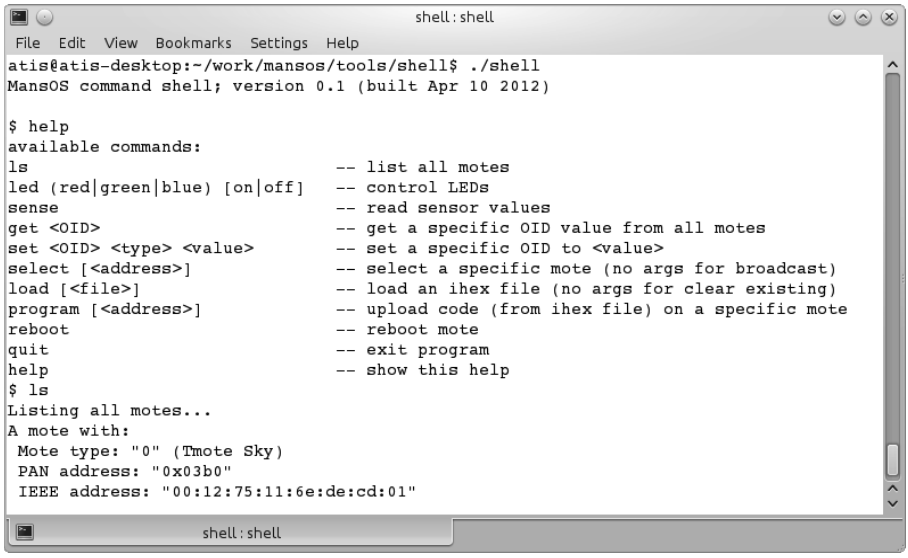


Fig. 4: MansOS shell

Reprogramming The need for run-time reprogramming support in WSN OS is apparent. For example, in our environmental monitoring use case, reprogramming just nine motes in outdoor conditions took more than two hours. Using over-the-air reprogramming reduces time requirements by an order of magnitude.

Run-time reprogramming in MansOS is performed in four stages:

1. binary code is read from file on disk and sent out to the WSN;
2. the code is transported through the network;
3. the code is received and stored on the target motes;
4. target motes reprogram themselves and run the received code.

The *first stage* is performed by MansOS shell (Fig. 4), to which a base station's mote is attached. For this purpose, the shell is extended with Intel IHEX file parser and multiple reprogramming related commands. For the *second stage*, SSMP is reused. A special object ID signalling binary data is used for code packets. The *third stage* is done by the reprogramming component of MansOS, which is included in application's binary image by specifying `USE_REPROGRAMMING=y` in its configuration file.

MansOS bootloader is responsible for the *final stage*, the most complicated one. The bootloader is another MansOS component which can be optionally included in application's binary file. If present, the bootloader is executed before any other MansOS code. If, before reboot, the reprogramming component has signalled the need to replace the existing program image, the bootloader performs the actual rewrite: it replaces MCU program memory contents with a new OS image taken from mote's external storage. For safety, use of "golden image" is supported. If bootloader detects that the system has failed to start multiple times in a row, it loads last usable OS image from the storage.

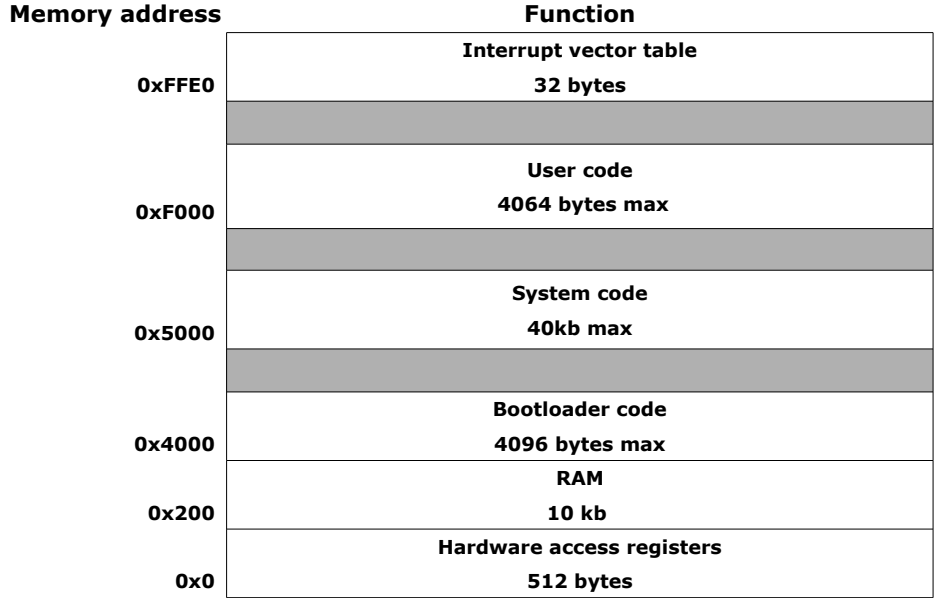


Fig. 5: MansOS memory layout of a *Tmote Sky* application compiled with runtime reprogramming support

Partial reprogramming is supported for energy-efficiency purposes. We observe that in WSN applications user code is smaller and require changes much more often than OS code. Therefore, it makes sense to separate system and user code, and allow to reprogram only one part without changing the other. In MansOS, partial reprogramming is implemented through address space separation of system and user code (Fig. 5). For user section only 4 KB of memory space is allocated, since the user code can be expected to be much smaller, as evidenced later in Table 2. For an even more striking example, `DemoRedLed` reprogramming demo application in MansOS is 14259 bytes long, of which only 56 bytes are user code. Therefore, avoiding system code reprogramming leads to great efficiency gains.

4.5 Usability and portability

Although ease-to-use is difficult to quantify, we nevertheless believe that it is an important property of WSN operating systems. Same goes for portability.

MansOS is multi-platform in the sense of supporting multiple hardware platforms (Tmote Sky, Arduino, Zolertia Z1 and more) and multiple architectures (MSP430 and Atmel AVR). MansOS provides platform-independent API for digital communication protocols (UART, SPI, I^2C) and MCU pin configuration. Therefore sensor, external memory and other peripheral drivers can be designed using platform-independent routines, allowing the same driver to be reused among multiple platforms and applications. The communication proto-

cols are provided in both hardware and software versions using unified API. The version to be used is selected at compile time, allowing to reuse peripheral drivers without modification.

MansOS is also multi-platform in the sense of multiple development host OS. While often neglected, quick and easy installation of a WSN OS is an important aspect of its usability. Therefore, MansOS provides self-executable Windows installer and a single Debian package as installation options.* This solution is more lightweight than using number of packages in different formats provided by TinyOS, or the virtual OS image provided by Contiki.

MansOS brings portability further: it is not tied to a particular platform, and also is not tied to a particular development environment. In particular, for MSP430 architecture multiple compilers are supported: *msp430-gcc* versions three and four, as well as IAR MSP430 compiler.

MSP430 GCC is a popular open-source solution for building MSP430 programs. It is powerful, full-featured compiler and can be easily integrated with the GNU debugger (GDB) and `make`. However, in contrast to its x86 version, the MSP430 code generator is not very mature. During development of MansOS we have met all kinds of problems: starting from MSP430 MCU hardware multiplication being used improperly, to generation of incorrect packet field access code, to a skipped instruction in the generated object file causing the program to abort, ending with linker optimizing the `main()` function away.

IAR Embedded Workbench IDE includes a commercial MSP430 C/C++ compiler and debugging options. Compared to open-source alternatives, it supports more MSP430 MCU models. So far we have had no problems with the code generation (possibly because IAR has been used much less than GCC). The main drawbacks of IAR compiler are the severe limitations of all evaluation editions.

In order to provide a visual programming option, we developed MansOS integrated development environment (IDE; Fig. 6). It supports visual and example-based programming, both frequently used by novice and embedded system programmers. The IDE is included in MansOS installation options, thus making the OS easily accessible even to inexperienced users. The IDE uses GCC as the default compiler, and has options for a single-click WSN mote programming, including remotely programming multiple motes at once.

5 Evaluation

This section describes the evaluation of application source code size, binary code and RAM usage, and resource usage by threads. MansOS is compared to three other operating systems: Contiki, TinyOS, and Mantis. Comparison with LiteOS is not performed, since the OS does not run on TelosB platform. We note that a brief evaluation of a few other aspects of MansOS was given in preceding sections.

* Available at `http://mansos.net`.

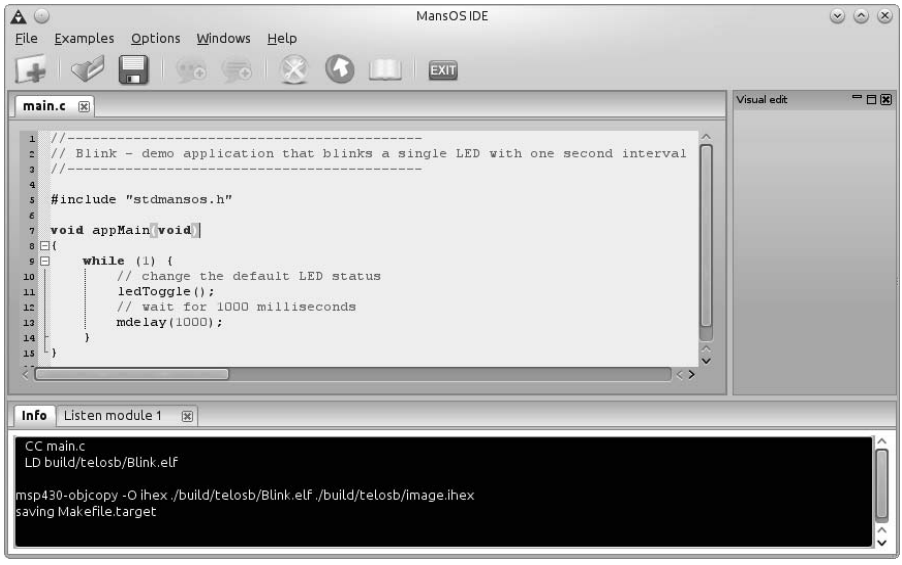


Fig. 6: MansOS IDE

5.1 Source code organization

For evaluation purposes four programs are implemented in MansOS, using both event-based and thread-based approach, as well as in Contiki, TinyOS and Mantis:

- *loop* – the simplest application: execute OS initialization code and then enter an endless loop;
- *radio tx* – transmit 100 byte radio packets periodically;
- *radio rx* – continuously listen for radio packets;
- *combined* – periodically sample sensors, toggle a LED, and transmit the sampled data to radio.

Source code for the *extended combined* application's event-based implementation in MansOS is given in Listing 2. The implementation is extended with external-flash logging.

First we compare source code size for the *combined* application in all five implementations (Fig. 7). The size is evaluated excluding comments and empty lines. Compared to other WSN OS, MansOS allows to write applications with the same functionality using shorter code. This is an important usability benefit of the system, because shorter code is more easy to understand and manage (at least when the complexity is the same). In contrast, large source codes size in TinyOS signal a potential usability problem with this OS. We point out that even though TinyOS applications are written in a different programming language (*nesC*), the abstraction level of the code is roughly the same: they are both high level languages. Further analysis is required to determine whether the complexity per line is small enough in TinyOS to balance out the additional code size.

Listing 2 Example MansOS application

```
#include <stdmansos.h>
#include <hil/extflash.h>
// define sampling period in miliseconds
#define SAMPLING_PERIOD 5000
// declare our packet structure
struct Packet_s {
    uint16_t voltage;
    uint16_t temperature;
};
typedef struct Packet_s Packet_t;
// declare a software timer
Alarm_t timer;
// declare flash address variable
uint32_t extFlashAddress;

// Timer callback function. The main work is done here.
void onTimer(void *param) {
    Packet_t packet;
    // turn on LED
    ledOn();
    // read MCU core voltage
    packet.voltage = adcRead(ADC_INTERNAL_VOLTAGE);
    // read internal temperature
    packet.temperature = adcRead(ADC_INTERNAL_TEMPERATURE);
    // send the packet to radio
    radioSend(&packet, sizeof(packet));
    // write the packet to flash
    extFlashWrite(extFlashAddress, &packet, sizeof(packet));
    extFlashAddress += sizeof(packet);
    // reschedule our alarm timer
    alarmSchedule(&timer, SAMPLING_PERIOD);
    // turn off LED
    ledOff();
}

// Application initialization
void appMain(void) {
    // wake up external flash chip
    extFlashWake();
    // prepare space for new records to be written
    extFlashBulkErase();
    // initialize and schedule our alarm timer
    alarmInit(&timer, onTimer, NULL);
    alarmSchedule(&timer, SAMPLING_PERIOD);
}
```

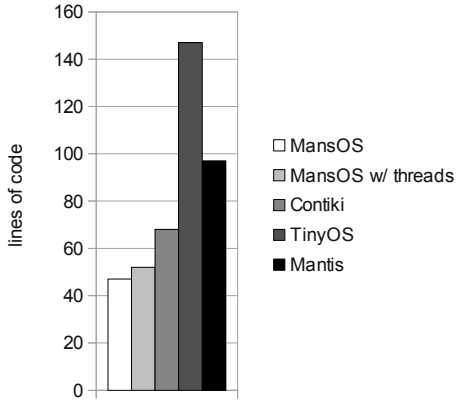


Fig. 7: Source code size comparison for the *combined* application (samples sensors and sends to radio)

5.2 Binary code size

Perhaps more important results are obtained by evaluating binary code sizes (Fig. 8). The source code is compiled for TelosB platform, using MSP430 GCC 4.5.3 compiler. For MansOS, `-O` optimization level is turned on (the default), since higher optimization levels historically have led to broken code. For other OS, their respective default optimization levels are used.

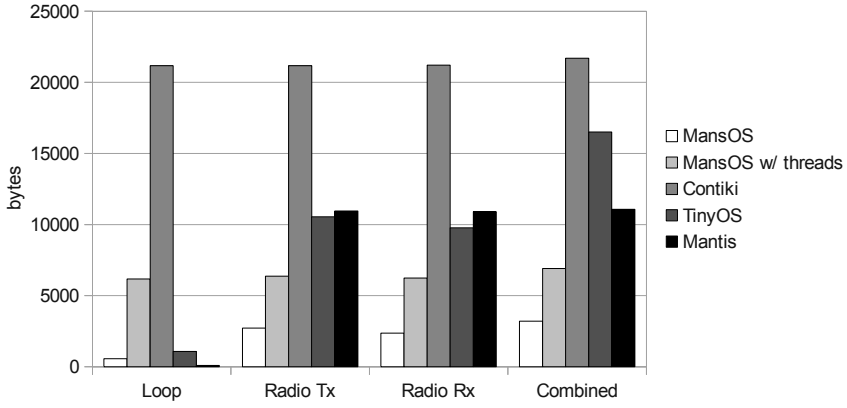


Fig. 8: Application binary size comparison for the *combined* application

MansOS shows the best results in three of four test cases, the only exception being *loop* program: in Mantis it uses only 102 bytes, compared to 566 bytes in MansOS (without threads).

Three of four WSN OS analysed try to reduce binary code size in some way. MansOS: by using the configuration mechanism, Mantis: by building separate components as libraries and linking them together, TinyOS: by topologically sorting all functions in source files and pruning unused ones from the final binary

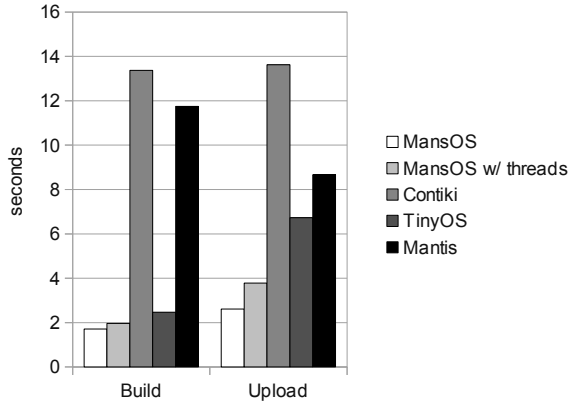


Fig. 9: Application upload time comparison for the *combined* application

code. Only Contiki pays no attention to this problem and demonstrates the worst results of all OS.

Larger binary code size in TinyOS are partially caused by limitations in this OS hardware abstraction model: direct access to radio chip's driver code is prohibited and Active Message interface has to be used.

As for Mantis, their approach is efficient, but suffers from usability problems. A number of changes are required to build their latest release with the current GNU compiler version, including defining `putchar()` as dummy function in user code and commenting out multiple references to `mos_led_display()` function in kernel code. The problems are caused by circular dependencies of the libraries. We can conclude that increasing the number of separately compiled components is detrimental to the usability of the core system, since the number of inter-component dependencies grows too fast.

Table 2: Flash memory usage in the *extended combined* application, bytes

	No threads	**With threads**
Radio	1394	1644
Kernel	702	720
Flash	454	454
USART & SPI	446	496
Arch & platform	308	370
ADC	182	182
User code	138	188
LEDs	38	38
Library routines	2	132
Threads	0	686
Total	**3270**	**4966**

Shorter binary code size means tangible benefits to the WSN OS user. Firstly, energy requirements in reprogramming are directly proportional to the code size,

Table 3: RAM usage in the *extended combined* application, bytes

	No threads	With threads
User code	14	4
Kernel	10	14
Radio	8	8
USART & SPI	8	8
Flash	2	2
ADC	2	2
Arch & platform	0	0
Threads	0	36
Total	**44**	**74**

if full reprogramming is used. Even though all OS allow some kind of partial reprogramming, full is still required when core parts of the system are changed. Secondly, smaller code leads to shorter development times, as putting the program on the mote becomes faster (Fig. 9). Furthermore, building MansOS programs is faster than their counterparts in other OS, because MansOS configuration mechanism excludes most of unnecessary source files from the build by default. TinyOS approach is efficient in this regard as well – we hypothesize it's because all *nesC* files are pre-compiled to a single C file for fast processing.

The MansOS in event-based form takes considerably less flash space than the threaded version. The difference is mostly due to the complexity of the thread implementation itself (Table 2). While using more resources in general, the threaded version leads to shorter user code and smaller RAM usage in it, because smaller state information has to be kept inside application's logic.

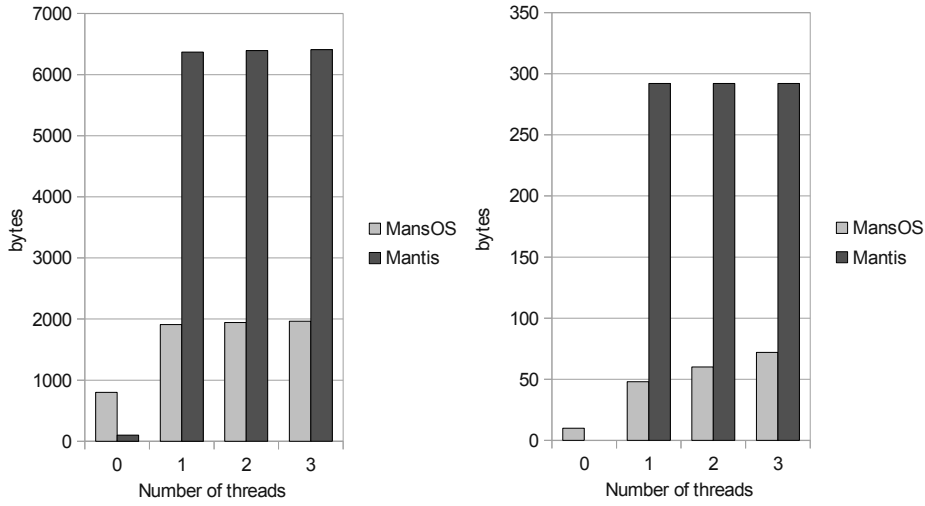
RAM usage is given without including memory allocated for stacks (256 bytes for each thread by default). Even though comparatively large amount of memory is used in this way, it would seldom cause problems for real applications, because code memory, not RAM, is the scarcest resource on Tmote Sky. This is evidenced by the example application (Table 2 and 3), because it uses proportionally more of total code memory (4966 bytes of 48 KB) than of total RAM ($74 + 256$ bytes of 10 KB).

5.3 Thread implementation

Thread implementation is compared with Mantis, because of the four operating systems considered only MansOS and Mantis include preemptive multithreading by default. Both include support for theoretically unlimited number of threads, although in MansOS the upper bound must be specified in compile time, and usually is small. The thread structure takes 12 to 16 bytes in RAM in MansOS (minimal and maximal configuration) and 21 bytes in Mantis.

Two distinctive features of MansOS thread implementation become apparent (Fig. 10):

- lower resource requirements. The scheduler used in MansOS is simpler, e.g. it has no separated queues for ready and sleeping threads. This trade-off is justified by the low number of threads typical in a WSN application,



(a) Flash usage **(b)** RAM usage

Fig. 10: Component usage comparison of a threaded application. *PB*: priority based scheduling, *RR*: round-robin based scheduling.

which allows the scheduler to process all threads in each context switch. Furthermore, in WSN applications the user threads can be expected to be cooperative, so the fairness of the scheduler is not a critical requirement. Finally, thread hierarchy in MansOS (user threads cannot preempt kernel) allows to reduce the number of locks required for thread-safe design.

- better adaptation. Mantis requirements are constant (the flash usage changes are due to longer user code), MansOS requirements are flexible and depend on the number of threads used;

For technical details it should be noted that Mantis example applications uses 128 byte stacks by default. Our selection of 256 byte stacks is motivated by large stack space requirements of library functions. For example, the `PRINTF` macro in MansOS eventually calls *libc* functions. The macro, when called without arguments, already uses 62 bytes of stack. The arguments passed to `PRINTF` can easily use ten or more bytes additionally. Therefore, future work includes implementing formatted print in the OS itself and in a more optimized fashion, as is done in Mantis. On the other hand, Mantis allocates stacks in heap, so more than the amount pictured (Fig. 10b) is used. Finally, we clarify that flash usage differences in Mantis are due to user code changes only.

The heart of the thread implementation in MansOS is the `schedule()` function that selects which thread to run next. The binary code size of this function depends on the number of threads used (Fig. 11). Round-robin based scheduling is more costly, partially because 32-bit *last-time-run* values are used instead of 16-bit priority values, and partially because thread's *last-time-run* is updated every time a thread is run, while priorities are kept unchanged during whole ex-

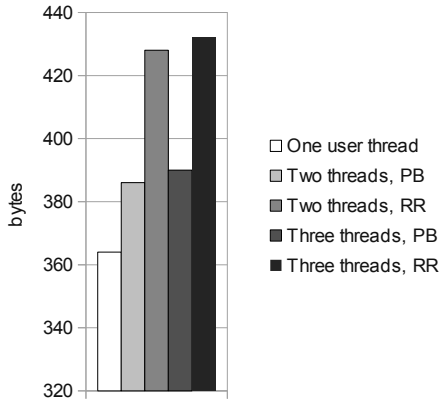
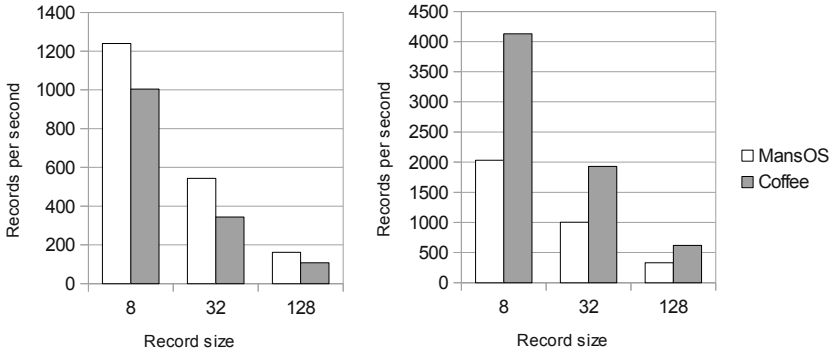


Fig. 11: Thread resource usage comparison: flash memory used by `schedule()` function.

ecution time. However, in all cases the space requirements can be easily satisfied by a typical WSN mote.

5.4 File system

The MansOS file system is evaluated against the Contiki file system, Coffee [15], because both of them have support for the TelosB platform's flash chip. Both are compiled with the GCC 4.6 compiler and the same optimization level.



(a) Write **(b)** Read

Fig. 12: File system write and read throughput

The most important function of a WSN node's storage system is the logging of sensor data. The data typically consist of constant-sized samples. To reflect this, the file systems are tested for how many data records of fixed size they can handle in a unit of time. The results for different record sizes are shown in Fig. 12a (write throughput) and Fig. 12b (read throughput).

The MansOS file system has better performance at writing data, but is slower than Coffee at reading data. Better write performance stems from the fact that

Coffee spends additional time relocating data as the file grows. The read performance of the MansOS file system could be impacted by additional logic layers that increase function call overhead. The read and write speed, at any rate, exceeds typical WSN requirements (from 1 to 100 measurements per second) by a far margin.

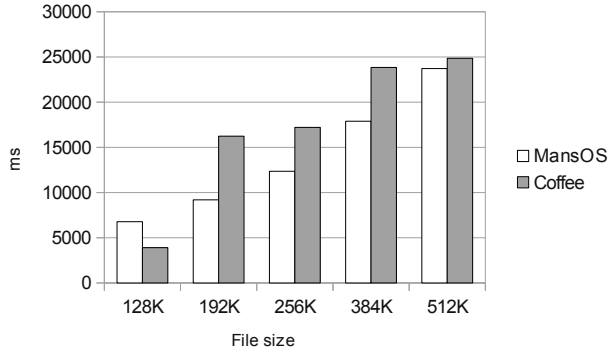


Fig. 13: File system write times

The file systems are also examined for the time it takes to write a file of a certain size: this shows how good the system is at allocating free space and handling files whose size is not known *a priori* (Fig. 13). The time taken by the MansOS file system scales linearly with size of files. Coffee demonstrates a more irregular pattern, which can be explained by that Coffee doubles the size of a file and copies its data once the file capacity is exceeded. Due to the same reason, Coffee cannot handle files that are larger than $\approx 66\%$ of the flash chip capacity (unless the planned file size is told before data are written to the file), while the MansOS file system allows allocating the whole chip to a single file.

6 Conclusions

We have described MansOS, a portable and easy-to-use operating system for wireless sensor networks and resource constrained embedded devices. MansOS is a feature-complete WSN OS with well-structured code. Compact binary code allows MansOS to avoid flash memory overuse problems that are especially prominent in Contiki.

Compared to LiteOS, MansOS is more portable, as it has logical separation between architecture and platform-specific code and the rest of the system.

Compared to Mantis, MansOS has lighter weight threads, as well as separation between the kernel thread and user threads, which in turn facilitates the design of the rest of the system. Locking is often not required, as user threads have no privilegies to preempt the kernel thread.

Compared to Contiki, MansOS is more modular, which in turn leads to lower resource usage overhead, as a MansOS application can use only those components it actually needs. The configuration system reduces both the number of

files that are compiled and the size of binary code, therefore usability is improved, as time taken to build and upload application becomes shorter. Furthermore, using MansOS means less run-time overhead, because module selection and function binding are done at compile time, not at execution time. Finally, MansOS provides a platform-independent (as much as possible) *implementation* of preemptive threads, complete with scheduler and thread-local variables, while Contiki gives only an *interface* of such a model.

Compared to TinyOS, MansOS is more approachable to users without WSN programming knowledge, especially if they are experienced in C programming, because MansOS includes support for multithreaded execution model and is written in plain C. Application source code tends to be significantly shorter as well, with no large obvious increase of complexity per code line, which means that programs written in MansOS are easier to understand and manage because of improved readability.

The OS is friendly to users regardless of their previous WSN programming experience. On one hand, MansOS is targeted towards novice users: it is easy to install, since Windows installer and Debian package are provided, and easy to start using, since MansOS IDE supports visual programming and example-based programming paradigms. On the other hand, MansOS offers a powerful and flexible configuration system for power users and developers of new hardware platforms. The system allows including and excluding specific features, even the MansOS kernel itself, thus providing seamless integration with existing user code and not forcing use of any resource-hungry features.

At the moment MansOS is evaluated in several environmental monitoring projects. In addition, the OS is used to teach WSN programming courses at University of Latvia as an alternative to TinyOS. Our future plans include more field tests of the OS, and support of new hardware platforms, including Intel 8051-compatible architectures.

Acknowledgements

We want to thank European Social Fund for financial support, grant Nr. 2009/-0219/1DP/1.1.1.2.0/APIA/VIAA/020. Work done by Janis Judvaitis on MansOS IDE and other parts of the system is gratefully acknowledged, as are all other MansOS developers.

References

1. Arduino, `http://arduino.cc/`
2. Atmel Corporation: Atmel AVR 8- and 32-bit Microcontrollers. `http://www.atmel.com/products/microcontrollers/avr/default.aspx`
3. Bhatti, S., Carlson, J., Dai, H., Deng, J., Rose, J., Sheth, A., Shucker, B., Gruenwald, C., Torgerson, A., Han, R.: MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms. Mobile Networks and Applications 10(4), 563–579 (2005)

4. Cao, Q., Abdelzaher, T., Stankovic, J., He, T.: The LiteOS Operating System: Towards Unix-Like Abstractions for Wireless Sensor Networks. In: IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks. pp. 233–244. IEEE Computer Society, Washington, DC, USA (2008)
5. Dai, H., Neufeld, M., Han, R.: ELF: An Efficient Log-Structured Flash File System. In: Proceedings of the 2nd Conference on Embedded Networked Sensor Systems (SenSys). pp. 176–187. ACM (2004)
6. Dunkels, A., Grönvall, B., Voigt, T.: Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. Local Computer Networks, Annual IEEE Conference on 0, 455–462 (2004)
7. Dunkels, A., Schmidt, O., Voigt, T., Ali, M.: Protothreads: simplifying event-driven programming of memory-constrained embedded systems. In: Proceedings of the 4th international conference on Embedded networked sensor systems. pp. 29–42. SenSys '06, ACM, New York, NY, USA (2006)
8. Durvy, M., Abeillé, J., Wetterwald, P., O'Flynn, C., Leverett, B., Gnoske, E., Vidales, M., Mulligan, G., Tsiftes, N., Finne, N., Dunkels, A.: Making sensor networks IPv6 ready. In: Proceedings of the 6th ACM conference on Embedded network sensor systems. pp. 421–422. SenSys '08, ACM, New York, NY, USA (2008)
9. Handziski, V., Polastre, J., Hauer, J., Sharp, C., Wolisz, A., Culler, D.: Flexible Hardware Abstraction for Wireless Sensor Networks. In: Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN 2005) (2005)
10. Klues, K., Liang, C., Paek, J., Musăloiu-e, R., Levis, P., Terzis, A., Govindan, R.: TOSThreads: Thread-Safe and Non-invasive Preemption in TinyOS. In: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09). pp. 127–140. ACM (2009)
11. Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., Culler, D.: Tinyos: An operating system for sensor networks. In: Ambient Intelligence. Springer Verlag (2004)
12. Libelium: Waspmote: The Sensor Device for Developers. `http://www.libelium.com/products/waspmote/`
13. Moteiv Corporation: Tmote Sky: Low Power Wireless Sensor Module. `http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf`
14. Texas Instruments: MSP430™ Ultra-Low Power 16-Bit Microcontrollers. `http://www.ti.com/msp430`
15. Tsiftes, N., Dunkels, A., He, Z., Voigt, T.: Enabling Large-Scale Storage in Sensor Networks with the Coffee File System. In: Proceedings of the 8th International Conference on Information Processing in Sensor Networks (IPSN). pp. 349–360 (2009)
16. Zolertia: Z1 Platform. `http://www.zolertia.com/ti`

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2012. Vol. 787
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 106–124 P.

Calculating The Layout For Dialog Windows Specified As Models

Sergejs Kozlovics

University of Latvia, Faculty of Computing
Raiņa blvd. 19, LV-1586, Rīga, Latvia
Institute of Mathematics and Computer Science, University of Latvia
Raiņa blvd. 29, LV-1459, Rīga, Latvia

sergejs.kozlovics@lumii.lv

Abstract In model-driven engineering, dialog windows can be described as models conforming to Dialog Metamodel. To make dialog models simpler, Dialog Metamodel does not require to specify exact coordinates and dimensions of graphical dialog elements. To transform a dialog model to a running dialog window a solution for laying out dialog elements is needed. We present an algorithm that calculates the layout by means of quadratic optimization.

Keywords: dialog metamodel, dialog engine, graphical user interface, GUI, layout, quadratic optimization.

1 Introduction

Graphical User Interface (GUI) is an essential part of many software products. There are numerous libraries for creating GUI dialogs and forms such as VCL[1], Windows::Forms[2], wxWidgets[3], GTK+[4], QT[5], Java Swing[6], etc.

[1] Visual Component Library. Available in Delphi and C++ Builder (currently maintained by CodeGear; previously maintained by Borland followed by Inprise) .

[2] Developed by Microsoft for the .NET framework. The open-source cross-platform .NET implementation Mono tries to re-implement Windows::Forms for various platforms.

[3] An open-source cross platform library providing "a truly native look and feel" for the applications.

[4] An open-source library used by the GNOME desktop environment in Linux.

[5] An open-source library used by the KDE desktop environment in Linux.

[6] A de facto GUI standard in Java. Developed by Sun, currently maintained by Oracle.

While GUI elements (buttons, input fields, check boxes, etc.) provided by these libraries are common, the libraries themselves are based on different conventions and have different APIs (Application Programming Interfaces). When a graphical dialog window is created directly using such a library, the developers need to know the API very well. Moreover, they have either to provide concrete coordinates for GUI elements (which may require certain calculations), or to specify their relative layout by library-specific facilities (e.g., by means of Java layout managers) requiring extra knowledge and skills. To help the developers, GUI designers were invented. With a GUI designer (such as Swing Designer, Qt Designer, etc.) dialog windows can be specified graphically just in a few clicks, and the GUI designer will generate the code for the corresponding GUI library.

Still, the benefits provided by GUI designers are not applicable to all cases. If the content of a dialog window (not only data, but also GUI elements) is computed at runtime, a GUI designer can be used to create only a carcass of a window; the content has to be filled up by means of the corresponding library API. Thus, the developer still needs a deep understanding of the GUI library and has to perform a non-trivial job of laying out GUI elements at runtime, especially when a complex window is being generated. Moreover, the traditional approach "generate GUI code→compile→run" is undesirable since the compiler may not be available on a user PC, and the generation process followed by launching a compiler or an interpreter may take a while.

In this paper we present a solution for creating graphical dialog windows automatically at runtime. The characteristic features of the proposed solution are:

- *The initial specification of a window is represented as a model.* We may think of the model as of abstract syntax of the sketch of a window on a sheet of paper. Some benefits of using a model are: independence on a particular GUI library or a particular programming language; the ability to create (generate) a model at runtime; the ability to specify dialog windows using concepts similar to concepts used in the sketch on a paper; support for model-driven development.
- *Developers need to specify only essential layout information in a model.* For instance, if the sizes of an input field and a button, as well as the spacing between them, have not been specified, these values will be chosen automatically in a way that contributes to a nice look of the resulting window.
- *A model is automatically transformed to a running dialog window.* The coordinates and unspecified dimensions of GUI elements are calculated automatically. When the window is being resized, dimensions and positions of elements will adapt to the window, preserving a nice look when possible.

In 2010 we have already published a metamodel[7] for specifying dialog windows [1]. In Section 2 we provide a brief summary of that Dialog Metamodel. The rest of the paper concentrates on the process of transforming a given dialog model to a running dialog window. We call the module that performs that process *Dialog Engine*[8]. In Section 3 we list certain agreements on layout information specified in a model and describe how Dialog Engine chooses the values for unspecified dimensions and positions of GUI elements. These automatically chosen values become a part of the given dialog model. In Section 4 we show how Dialog Engine transforms this (adjusted) dialog model to an input of the quadratic optimization solver. The solver returns the coordinates of GUI elements, and Dialog Engine displays the window. Section 5 is devoted to related work, and Section 6 concludes the paper.

2 A Glance At Dialog Metamodel

Fig. 1 depicts Dialog Metamodel (we published this metamodel in *Acta Universitatis Latviensis* in 2010 [1]). Each GUI element is either a separate component (such as a button, an input field, etc.; see class *Component* and its subclasses on the left in Fig. 1), or a container (a component containing other components; see class *Container*). Containers and components logically form a tree structure (see the composition between *Component* and *Container* in Fig. 1): each container has an ordered list of components lying within it. The layout of child components is determined by the type of the container. We distinguish 13 container types encircled by the rounded rectangle on the right in Fig. 1. Table 1 explains these container types. These are invisible containers used to construct the structure of a dialog window. The detailed explanation of other (visible) containers as well as of particular GUI components can be found in the paper mentioned above [1].

The initial dialog window (represented by the class *Form*) is a vertical box by default, but it can be transformed to a container of any other type (say, *HorizontalBox*) by adding the corresponding container as the only child, and putting all other components inside that child.

Dialog Metamodel contains also special classes called *events* (rounded rectangles in Fig. 1) and *commands* (ellipses in Fig. 1). These classes are used to ensure communication between Dialog Engine and other modules (usually, model transformations; hereinafter we refer to these modules as transformations). Transformations create commands for Dialog Engine, and Dialog Engine creates events that can be handled by transformations. Events and commands are temporary objects — they are deleted just after the desired action has been performed.

[7] A metamodel is a language for specifying models.
[8] That is why Dialog Metamodel is also called Dialog Engine Metamodel.

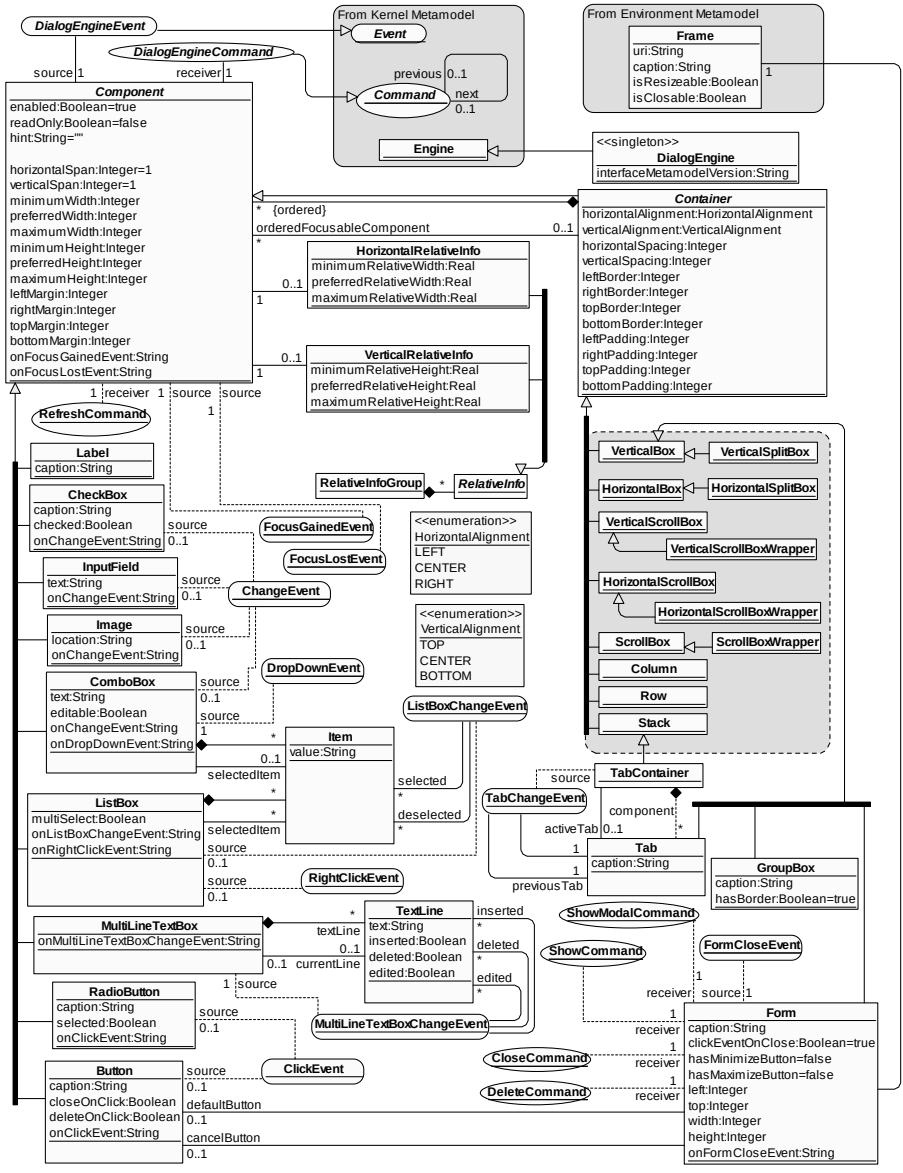


Fig. 1: Dialog Metamodel.

Table 1: Laying out child components in different container types.

Container Type	**The layout of child components**
VerticalBox	Vertical, components are laid one beyond another.
VerticalSplitBox	Components are laid one beyond another, but visible splitters are inserted between components. Each splitter can be moved by the user to enlarge a component on one side and to lessen a component on the other side.
HorizonalBox	Horizontal, components are laid one after another.
HorizontalSplitBox	Similar to *VerticalSplitBox*, but components are laid horizontally, and splitters are vertical lines.
VerticalScrollBox	Similar to *HorizontalBox*, but with a vertical scrollbar. Components are laid out horizontally one after another until the border of a scrollbar is reached. The remaining components continue at the next "row".
VerticalScrollBoxWrapper	Can be used to add a vertical scroll bar for a component without scrollbars. Technically, the same as *VerticalScrollBox*, but the default values for layout information differ to provide better look.
HorizontalScrollBox	Elements are laid out like columns in a newspaper. "Columns" can be scrolled horizontally by means of a horizontal scrollbar.
HorizontalScrollBoxWrapper	Can be used to add a horizontal scroll bar for a component without scrollbars.
ScrollBox	The same as *VerticalBox*, but one or two scrollbars may appear, when the visible part of this scroll box is not able to accommodate all the children. Can be turned into a *HorizontalBox* by adding a *HorizontalBox* as a child.
ScrollBoxWrapper	Can be used to scroll horizontally and vertically a component without scrollbars.
Column	Similar to *VerticalBox*, but components inside two neighbouring columns are aligned to form a table-like structure (have the same y-coordinates).
Row	Similar to *HorizontalBox*, but components inside two neighbouring rows are aligned (have the same x-coordinates).
Stack	Components are laid out like cards, occupying the same space on a dialog window. Used to implement tabs (see *TabContainer*).

When a model transformation needs to show a dialog window for a given dialog model, it creates either a *ShowModalCommand*, or a *ShowCommand* instance depending on whether the dialog needs to be modal or modeless. The control is passed to Dialog Engine, which displays the corresponding window on the screen. Execution of a *ShowModalCommand* finishes only after the window is closed. On the contrary, *ShowCommand* returns immediately after displaying the window, which remains visible while the transformation continues.

When a dialog window is active, certain events (usually, corresponding to user clicks and keystrokes) can occur. On each such event, Dialog Engine creates some *Event* instance. In Dialog Metamodel (Fig. 1), attributes starting with "on" (e.g., *onClickEvent*) are used to specify model transformations that Dialog Engine will call on the corresponding events. After the transformation processes the event, the event instance is deleted. When processing events, transformations can issue new commands to Dialog Engine, e.g., a command to show another dialog, or a *RefreshCommand* to refresh an already running dialog or its part (a subtree rooted at the given component) when some changes have been made to the corresponding dialog model.[9]

On certain events (e.g., when processing a *ClickEvent* of some "Close" button, or on a *FormCloseEvent*, which occurs when the user clicks the "X" button), a transformation can issue a *CloseCommand* to ask Dialog Engine to close an active dialog window. If the dialog is modal, *CloseCommand* also indicates that execution of the corresponding *ShowModalCommand* must finish.

After closing the dialog window, a *DeleteCommand* can be used to delete a dialog model that is not needed any more.

In this paper we focus at layout information common to all GUI components, even to components that may be added later. This layout information is found in classes *Component* (excluding the first three and the last two attributes), *HorizontalRelativeInfo*, *VerticalRelativeInfo*, and *Container*. The next section explains what do these attributes mean[10] and how to choose the unspecified values of these attributes.

3 Specifying The Layout Of GUI Elements

The *horizontalSpan* and *verticalSpan* attributes of the Component class are used in rows and columns, where a component needs to span multiple cells. Without these two attributes, it could be impossible to lay out containers as depicted in Fig. 2(a). Fig. 2(b) shows how this structure can be specified with the appropriate values of *horizontalSpan* and *verticalSpan*. The default value for both these attributes is 1 meaning that a component occupies a single cell, when placed in a row or column container.

[9] Thus, a call stack similar to function call stack can occur.

[10] We repeat some relevant information published earlier [1].

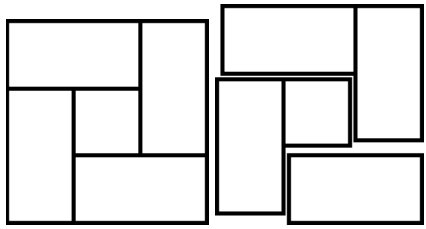


Fig. 2: (a) An example of five containers that cannot be laid out using horizontal and vertical boxes only. (b) The arrangement of the same five containers using rows. The first row contains two components: the first one spans two columns (*horizontalSpan*=2, *verticalSpan*=1), and the second one spans two rows (*horizontalSpan*=1, *verticalSpan*=2). The first component of the second row spans two rows (*horizontalSpan*=1, *verticalSpan*=2); neither rows nor columns are spanned by the second component (*horizontalSpan*=1, *verticalSpan*=1). The third row has only one component that spans two columns (*horizontalSpan*=2, *verticalSpan*=1).

The *minimumWidth* and *minimumHeight* attributes specify strict lower bounds (in pixels) for the width and height, respectively. The *preferredWidth* and *preferredHeight* attributes specify the preferred width and height. These values are not strict: when preferred dimensions of a component conflict with other, more important, constraints, deviations from the preferred dimensions are allowed. The *maximumWidth* and *maximumHeight* attributes specify the upper bound for the dimensions. These constraints are considered strict, but with the following exception: when other strict constraints are unsatisfiable, maximum width and height are allowed to increase by a minimal value to satisfy those constraints. The reason for introducing non-strict constraints relies on the following principle: if all the constraints are unsatisfiable, it is better to show a dialog window that violates some non-important layout constraints than to throw an exception and leave the user without the dialog window at all.

> *Example.* If a button has *minimumWidth* set to 100, but *maximumWidth* set to 0, the actual width of the button will be 100, i.e., the maximum width will increase by 100 pixels — the minimal value that satisfies the *minimumWidth* constraint. In this case, the *preferredWidth* value does not matter, since the width is determined by more strict minimum and maximum constraints.

The meaning of attributes for specifying margins, borders, spacings, and paddings (from classes *Component* and *Container*) is depicted in Fig. 3 (only horizontal values are depicted; vertical values have similar meaning).

The semantics of Dialog Metamodel states that there exists *gravity* between each container and its child components. That means that children edges tend to "stick" to the corresponding parent edges. However, if a child component

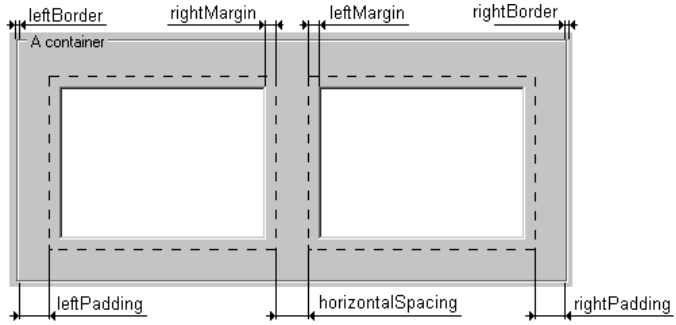


Fig. 3: An example illustrating what do values for margins, borders, paddings and spacings mean.

reaches its *maximumWidth* or *maximumHeight* constraint, the gravity does not work any more. In this case we have to specify how such children have to be put within the parent. The *horizontalAlignment* and *verticalAlignment* attributes of the *Container* class come to aid here. These values specify the horizontal alignment (left, center, or right) as well as vertical alignment (top, center, or bottom) for child components when they cannot not be stretched (by means of "gravity") any more.

Not only absolute, but also relative dimensions are supported by Dialog Metamodel. Relative widths and heights that relate to each other are grouped (see class *RelativeInfoGroup*). For example, to specify that widths of some three components have ratio 2:3:4, we attach three *HorizontalRelativeInfo* instances, one to each component, and set the values of *preferredRelativeWidth* to 2, 3, and 4, respectively. Finally, we attach all the three *HorizontalRelativeInfo* instances to a *RelativeInfoGroup* instance. Note that there is no need to specify more than one *HorizontalRelativeInfo* and more than one *VerticalRelativeInfo* for each component: if a particular width or height of some component appears in several groups, then these groups depend on each other and may be replaced by a single group by adjusting the ratio.

The minimum and maximum relative widths and heights are useful to control relative dimensions of components, when the dialog is being resized. An example is given in Fig. 4. The button 1 is not resizeable, and the preferred width ratio for both buttons is 1:1. If the user resizes the form, and the preferred ratio 1:1 could not be met, the button 2 is allowed to be up to two times wider (*maximumRelativeWidth*=2) or shorter (*minimumRelativeWidth*=0.5) than the button 1.

Like preferred absolute dimensions, preferred relative dimensions are not strict. However, minimum and maximum relative sizes are strict, but they are used in cooperation with the corresponding absolute sizes: having both relative and absolute minimum and maximum bounds, we can make absolute bounds

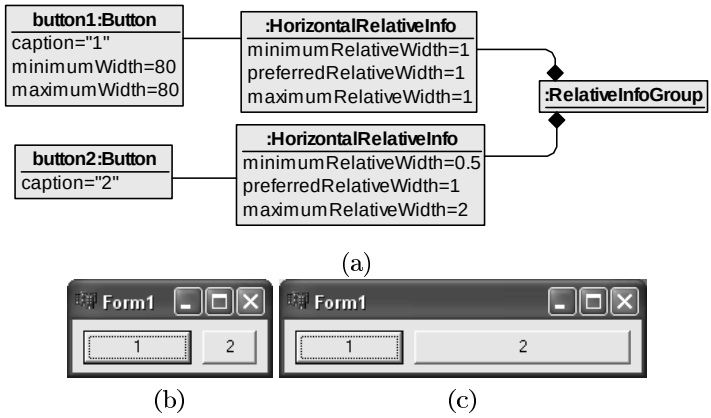


Fig. 4: An instance (a) demonstrating the usage of minimum and maximum relative sizes. The minimum (b) and the maximum (c) sizes of button 2.

more strict and forget about the relative bounds (see the next section for details).

Specifying all the constraints and sizes from above forces the developer to think more on layout than on the content of a dialog window. Thus, we allow the developer to leave all values unspecified, while preserving the possibility to specify important values (or all, if all are important). A question arises: how to choose values, if they have not been specified? The answer depends on a particular component or container. Table 2 lists some examples. The principles shown in Table 2 can be applied, when choosing values for other GUI elements.

4 Using Quadratic Optimization To Obtain Coordinates

This section explains how, given a dialog instance specified according to Dialog Metamodel, the quadratic optimization can be used to lay out GUI components in that dialog window.

4.1 The QMDC and the Extended QMDC Problems

The problem of quadratic minimization subject to difference constraints (QMDC) is as follows. Given n variables $x_0, x_2, \ldots, x_{n-1}$, minimize the quadratic function

$$\sum_{0 \le i < n} a_i x_i^2 + \sum_{0 \le i < j < n} b_{ij} x_i x_j + \sum_{0 \le i < n} c_i x_i$$

subject to difference constraints

$$x_i - x_j \ge d_{ij}, \text{ where } 0 \le i, j < n.$$

Table 2: How Dialog Engine chooses values for unspecified layout attributes.

Component or container	**Values**
Button (a non-resizeable component)	minimumWidth=preferredWidth=maximumWidth =TextWidth(caption)+c_1; minimumHeight=preferredHeight=maximumHeight =TextHeight(caption)+c_2; all margins=0; for all buttons within the same container: all relative sizes set to 1 (this ensures that buttons within the same container have equal sizes regardless their labels)
InputField (horizontally resizeable, but vertically non-resizable)	minimumWidth=c_3; preferredWidth=c_4; maximumWidth=∞; minimumHeight=preferredHeight=maximumHeight =TextHeight(text)+c_5; all margins=0

HorizontalBox (resizeable container with no visible borders and zero paddings; horizontal layout of children)	minimumWidth=minimumHeight=0;
	maximumWidth=maximumHeight=∞;
	all margins=0;
	horizontalAlignment=verticalAlignment=CENTER;
	horizontalSpacing=c_6; verticalSpacing is not applicable;
	all borders=0;
	all paddings=0;

GroupBox (a visible container with borders and padding; vertical layout of children)	minimumWidth=TextWidth(caption)+c_7;
	minimumHeight=TextHeight(caption)+c_8;
	maximumWidth=maximumHeight=∞
	all margins=0;
	horizontalAlignment=verticalAlignment=CENTER;
	horizontalSpacing is not applicable; verticalSpacing=c_9;
	leftBorder=c_{10}; rightBorder=c_{11};
	topBorder=TextHeight(caption)+c_{12}; bottomBorder=c_{13};
	leftPadding=c_{14}; rightPadding=c_{15}; topPadding=c_{16};
	bottomPadding=c_{17};

Note 1. If preferred size for a container has not been specified, Dialog Engine is allowed to leave that value unspecified; in this case this preferred value will not be considered by quadratic optimization, and the preferred size of the container will be determined by inner components. For non-containers (final components such as a button or an input field) preferred sizes still have to be chosen by Dialog Engine.

Note 2. Constants c_i depend on the GUI library.

Note 3. The infinity constant∞ is the absolute maximum size for any GUI component (e.g., 10 000 pixels). If some size reaches ∞, the component becomes so huge that it is unreasonable to show it to the user, so Dialog Engine throws an exception.

If also the constraints in the form

$$x_i - x_j \geq d_{ij} \geq m_{ij}, \tag{1}$$

are allowed, then we get the Extended QMDC problem (EQMDC). Here d_{ij} are the desired values and m_{ij} are the minimum values. In case the constraints taking into a consideration only d_{ij} are unsatisfiable, one or more of d_{ij} values may be decreased preserving $d_{ij} \geq m_{ij}$, i.e., d_{ij} cannot be decreased by more than by $d_{ij} - m_{ij}$.

4.2 The application of EQMDC

This section explains how to transform a dialog instance to the input of the EQMDC problem. The EQMDC solver is described in Section 4.7.

The variables we need are as follows. For each component C four variables are introduced to specify the left, right, top and bottom coordinates: x^C_{L}, x^C_{R}, y^C_{T} and y^C_{B}. The variables that bound the component with its margins are $x^C_{\mathrm{L}M}$, $x^C_{\mathrm{R}M}$, $y^C_{\mathrm{T}M}$ and $y^C_{\mathrm{B}M}$. If C is a container, then the variables for storing C bounds without its border are $x^C_{\mathrm{L}B}$, $x^C_{\mathrm{R}B}$, $y^C_{\mathrm{T}B}$ and $y^C_{\mathrm{B}B}$. Finally, the variables for bounding the inner area of C (without padding) are $x^C_{\mathrm{L}P}$, $x^C_{\mathrm{R}P}$, $y^C_{\mathrm{T}P}$, $y^C_{\mathrm{B}P}$.

The following subsections explain which constraints and terms to be minimized are introduced for 1) absolute sizes; 2) relative sizes; 3) margins, borders, padding and spacing, and 4) gravity and alignment. We provide the constraints for the x-dimension only since the constraints for the y-dimension are similar.

4.3 Constraints and minimization terms for absolute sizes

The constraint concerning minimum width, obviously, is:

$$x^C_{\mathrm{R}} - x^C_{\mathrm{L}} \geq \mathit{minimumWidth};$$

But the constraint for the maximum width is being written in the extended form:

$$\begin{aligned} x^C_{\mathrm{L}} - x^C_{\mathrm{R}} &\geq -\mathit{maximumWidth} \\ &\geq -\infty. \end{aligned}$$

This is the same as

$$\begin{aligned} x^C_{\mathrm{R}} - x^C_{\mathrm{L}} &\leq \mathit{maximumWidth} \\ &\leq \infty, \end{aligned}$$

but written in the form of (1).

The preferred sizes are specified not by means of constraints, but as terms of the function to be minimized. We add to the minimization the term

$$c \cdot \left((x^C_{\mathrm{R}} - x^C_{\mathrm{L}}) - \mathit{preferredWidth} \right)^2.$$

Here the penalty for the actual width $(x^C_{\mathrm{R}} - x^C_{\mathrm{L}})$ to be distinct from the preferred width grows quadratically. This ensures that even the component cannot have

the preferred size (due to constraints), the actual size will be close to the preferred. The constant c may be determined in the experimental way taking into a consideration other terms to be minimized.

4.4 The minimization terms for relative sizes

Figure 5 shows that specifying relative sizes may easily lead to unsatisfiability.

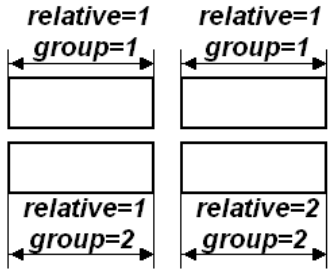


Fig. 5: An example of the unsatisfiable constrains on relative sizes. Two rows, each consisting of two components. The word *relative* may refer to any of the minimum/preferred/maximum relative width fields. The word *group* refers to the relative width group. If all these components have also some positive *minimumWidth* values, then the constraints on relative sizes cannot be satisfied.

There are two rows, each consisting of two components. The component sizes should be aligned to grid. On the one hand, all relative widths (minimum, preferred and maximum) for the first two components are set to 1, i.e., the widths of the first row components should be equal. On the other hand, the second component in the second row must be two times wider than the first component in the same row. Obviously, the only solution is when all the components have zero widths. But specifying some positive values for *minimumWidth* fields of these components gives us no solution at all. To make the layout engine flexible, we use the method described below that will find an approximate solution should such a situation as in Figure 5 occur.

Assume there are two components, A and B, in the same relative width group (for the heights the method is similar). Let them have preferred relative widths r_1 and r_2, and let x^A_{L} and x^A_{R} be variables for the left and the right bounds of the first component as well as x^B_{L} and x^B_{R} for the second. Then, obviously, the desired equation is:

$$r_2 \cdot (x^A_{\mathrm{R}} - x^A_{\mathrm{L}}) = r_1 \cdot (x^B_{\mathrm{R}} - x^B_{\mathrm{L}}). \tag{2}$$

Since not always this equation may be satisfied by the reasons mentioned above, it is better to replace it with the approximate equation. Let's write (2) in this

form:

$$r_2 \cdot (x_{\mathrm{R}}^A - x_{\mathrm{L}}^A) - r_1 \cdot (x_{\mathrm{R}}^B - x_{\mathrm{L}}^B) \approx 0.$$

But this form may be rewritten as a term to be minimized by quadratic optimization algorithm:

$$\left(r_2 \cdot (x_{\mathrm{R}}^A - x_{\mathrm{L}}^A) - r_1 \cdot (x_{\mathrm{R}}^B - x_{\mathrm{L}}^B)\right)^2. \tag{3}$$

In case all the constraints can be satisfied, this term will be zero, and, thus, the desired relation will be hold. Otherwise, the difference between the desired and the actual relation will tend to be zero.

Since for any positive k the relative widths $k \cdot r_1$ and $k \cdot r_2$ denote the same relation as r_1 and r_2, we want the quadratic term (3) also be the same. So, prior to creating the term (3), we use the assignment

$$\begin{pmatrix} r_1 \\ r_2 \end{pmatrix} \leftarrow \begin{pmatrix} \frac{r_1}{r_1+r_2} \\ \frac{r_2}{r_1+r_2} \end{pmatrix}$$

for the normalization.

If minimum/maximum relative sizes are also specified, then they are used as follows. Let $(C_1, r_1, s_1), (C_2, r_2, s_2), \ldots, (C_n, r_n, s_n)$ be the triples where r_i denote minimum relative widths and/or heights contained in the same group. C_i are the corresponding components and s_i are equal to corresponding (absolute) minimum width and/or height values.

A coefficient k is calculated first:

$$k \leftarrow \max\left\{\frac{s_i}{r_i}\right\}.$$

Then, for each C_i we set the corresponding *minimumWidth* or *minimumHeight* to the value $k \cdot r_i$.

The same refers to maximum sizes with the following differences:

- For all the triples (C_i, r_i, s_i) with the r_i set to maximum relative width/height and s_i set to maximum width/height, r_i and s_i must be defined.
- To calculate k we use not max, but $\min\{s_i/r_i\}$.

4.5 Constraints for margins, borders, padding and spacing

We introduce the following constrains for the margins:

$$x_{\mathrm{L}}^C - x_{\mathrm{L}M}^C \geq \textit{leftMargin},$$
$$x_{\mathrm{R}M}^C - x_{\mathrm{R}}^C \geq \textit{rightMargin}.$$

If C is a container, the borders are specified as

$$x_{\mathrm{L}B}^C - x_{\mathrm{L}}^C \geq \textit{leftBorder},$$
$$x_{\mathrm{R}}^C - x_{\mathrm{R}B}^C \geq \textit{rightBorder}.$$

Finally, if C is a non-scrollable container, then we add the following constraints for padding:

$$x^C_{\mathrm{L}P} - x^C_{\mathrm{L}B} \geq \mathit{leftPadding},$$
$$x^C_{\mathrm{R}B} - x^C_{\mathrm{R}P} \geq \mathit{rightPadding}.$$

In case of a scrollable container the second constraint is not added.

The spacing between two components A and B (A on the left of B), obviously, is introduced by the constraint

$$x^B_{\mathrm{L}M} - x^A_{\mathrm{R}M} = \mathit{horizontalSpacing}$$

(equation can be replaced by two inequalities).

4.6 Gravity and alignment

Assume there are nested components (see Figure 6).

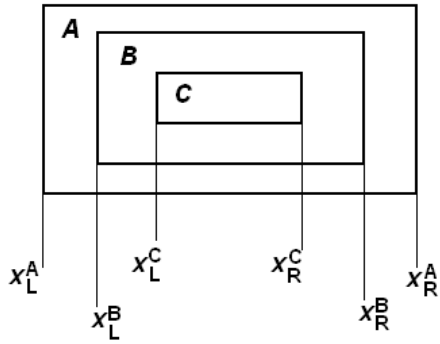


Fig. 6: The nested components. There exists gravity between the borders that forces the inner component to be resized (unless it has the maximum size specified) when the outer component is resized.

If the component C is not resizeable, we still want the component B to be resizeable along with component A. Thus, the gravity between components B and C should be less than between A and B.

Referring to Figure 6, gravity tends to minimize the following differences: $x^C_{\mathrm{L}} - x^B_{\mathrm{L}}$, $x^B_{\mathrm{R}} - x^C_{\mathrm{R}}$, $x^B_{\mathrm{L}} - x^A_{\mathrm{L}}$ and $x^A_{\mathrm{R}} - x^B_{\mathrm{R}}$. The first two differences must be "weaker". That is, if C is not resizeable, then there should be no gravity between B and C. In order to achieve this, we use the following *linear* terms to be minimized:

$$k \cdot (x^C_{\mathrm{L}} - x^B_{\mathrm{L}}) + k \cdot (x^B_{\mathrm{R}} - x^C_{\mathrm{R}}) + l \cdot (x^B_{\mathrm{L}} - x^A_{\mathrm{L}}) + l \cdot (x^A_{\mathrm{R}} - x^B_{\mathrm{R}}),$$

where $k < l$. Assume C is not resizeable, and A has just been stretched. That means we have fixed the sum $(x_{\mathrm{L}}^C - x_{\mathrm{L}}^A) + (x_{\mathrm{R}}^A - x_{\mathrm{R}}^C)$. If $k < l$, then the sum of the four terms will be minimal when the last two terms are zero. That is also the desired behaviour for the gravity. If B is a vertical box with n children, the "weight" k should be divided by n since the minimization terms added by all children sum up.

We should not forget to specify also

$$\begin{aligned} x_{\mathrm{L}}^C - x_{\mathrm{L}}^B &\geq 0, \\ x_{\mathrm{R}}^B - x_{\mathrm{R}}^C &\geq 0, \\ x_{\mathrm{L}}^B - x_{\mathrm{L}}^A &\geq 0, \\ x_{\mathrm{R}}^A - x_{\mathrm{R}}^B &\geq 0. \end{aligned}$$

If the children have to be left-aligned, then instead of gravity between the left container border and the first child we introduce the constraint

$$x_{\mathrm{L}}^B - x_{\mathrm{L}}^A = 0,$$

where A is a container and B is the left child. The same is when the children have to be right-aligned. However, if the children are to be centered, we add the term

$$c \cdot \left((x_{\mathrm{L}}^B - x_{\mathrm{L}}^A) - (x_{\mathrm{R}}^A - x_{\mathrm{R}}^B)\right)^2$$

to the minimization that is used to make the distances from the component to the left and right borders of the parent equal. We add also the constraints

$$\begin{aligned} x_{\mathrm{L}}^B - x_{\mathrm{L}}^A &\geq 0, \\ x_{\mathrm{R}}^A - x_{\mathrm{R}}^B &\geq 0. \end{aligned}$$

4.7 An EQMDC Solver

There exists a method for solving QMDC in a moderate time by a technique that is based on the projective gradient method [2]. The Extended QMDC problem can be reduced to the ordinary QMDC in the following way. First, a constraint graph $G = (V, E)$ is created [3, Section "Difference constraints and shortest paths"]. Here $V = \{s, v_0, v_1, \ldots, v_{n-1}\}$, where all v_i correspond to variables x_i and s is a special start vertex. Edge set E is

$$E = \{(v_i, v_j) : x_i - x_j \geq d_{ij} \geq m_{ij} \text{ is a constraint}\}$$

$$\cup\{(s, v_0), (s, v_1), \ldots, (s, v_{n-1})\}.$$

In the beginning, we consider only d_{ij} values. Rewriting (1), we have:

$$x_j - x_i \leq -d_{ij}.$$

Then, we assign the weight of the edge (v_i, v_j) to the value $-d_{ij}$, while the edges (s, v_i) have the weight 0.

Now, if G does not contain a negative cycle, then the system is solvable, and m_{ij} can be removed leaving only d_{ij}. If G does contain a negative cycle, then the weights of the edges in the cycle are increased to meet the constraints on m_{ij} (corresponding d_{ij} values are *decreased*). If the cycle cannot be eliminated, there is no solution. Otherwise, we continue until all the negative cycles are eliminated.

To achieve practically good execution time, we use the Bellman-Ford-Tarjan algorithm with the subtree disassembly method for finding the negative cycles. Since for directed acyclic graphs negative cycle detection can be performed in linear time, the strongly-connected components are searched in advance. So, the non-linear Bellman-Ford-Tarjan algorithm (which has the upper bound $O(V^3)$) is executed only on strongly-connected components, while considering the edges between these components takes linear time as these edges do not form a cycle.

5 Related Work

Since there exist algorithms for user interface layout that use linear constraints [4,5], one may be interested why the quadratic (and not linear, or, possible, cubic) optimization is preferable here. The two reasons can be mentioned: 1) it is impossible to implement some constraints (like constraints for preferred sizes) by means of a linear function only, and 2) optimizing other non-linear (cubic et al.) functions can be very time-consuming. Regarding the implementation of QMDC, any method can be used here (while we use the method by Freivalds and Ķikusts [2], one could use the method by Hosobe, for instance [6]).

It can be proven that expressive power of Dialog Metamodel with predefined types of containers is at least the same as of standard Java layout managers from the Java Swing library, including a comprehensive *GroupLayout* manager. The main idea of the proof is to show how each standard Java Swing layout manager can be simulated by means of the container types provided by Dialog Metamodel.

There is an interesting difference in resizing policy between our approach and the approach used by the QT library [7]. In QT, when a layout (such as vertical, horizontal, or grid layout) for a container is set, components inside this container are resized by default. To prevent resizing, special components called horizontal and vertical spacers can be used. Spacers act as springs that produce a counterforce for resizing. In contrast, our approach uses maximum width and height constraints to prevent resizing.

Model-driven graphical tool building platforms such as Eclipse GMF [8,9], Microsoft DSL Tools [10], Metaclipse+ [11], and others, usually provide a standard mechanism for creating dialog boxes such as property editors. These stand-

ard mechanisms permit only limited customizations of a dialog box (e.g., we can specify the names of properties and their values, but we cannot add some extra buttons). While the expressiveness of such simple dialogs is sufficient for most cases, some tools (like Microsoft DSL Tools) permit also specifying dialogs of arbitrary complexity in some object-oriented programming language (e.g. C#). But the additional knowledge and skills are required here. Moreover, the model-driven approach is lost.

6 Conclusion

Dialog Engine, mentioned in this paper, has been implemented in C++ Builder, and currently is used in tools based on the Transformation-Driven Architecture, TDA. Examples of such tools are a graphical ontology editor OWLGrEd [12] and a UML editor GradeTwo [13]. Quadratic optimization is called each time when a dialog window is being displayed on the screen, when the user resizes a dialog window, and when windows are changed and refreshed at runtime (by issuing a *RefreshCommand* from Dialog Metamodel).

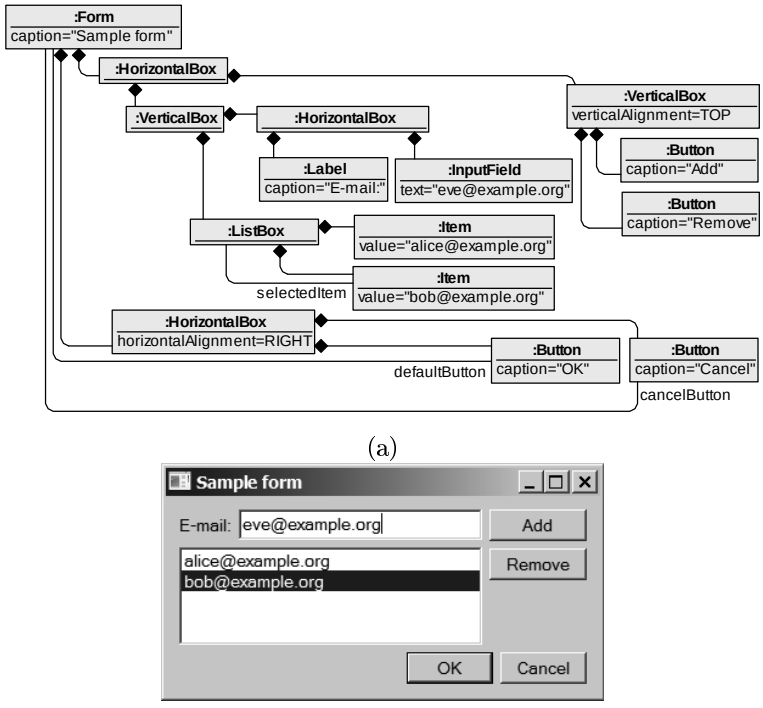


Fig. 7: (a) A model of a sample form. (b) The resulting dialog window on the screen.

An example of a dialog model that does not specify any coordinates and dimensions (only containers and some alignments) is depicted in Fig. 7(a). The resulting dialog window, produced by Dialog Engine, is depicted in Fig. 7(b). This is a real screenshot.

As noticed by Dmitrijs Logvinovs, the number of variables used in quadratic optimization can be reduced up to two times by combining them (e.g., variables for margins can be combined with the corresponding component coordinates). He also started a Java implementation that reduces the number of variables and takes an advantage of using Java reflection mechanism for loading GUI elements at runtime.

Having Dialog Metamodel and a working Dialog Engine for it, Dialog Metamodel can be used as a formal language for describing abstract syntax of dialog windows.

Acknowledgements

I thank Kārlis Freivalds for valuable personal conversations on quadratic optimization. I thank also Dmitrijs Logvinovs for certain improvements that he noticed and incorporated in the Java version of Dialog Engine.

This work has been partially supported by the European Social Fund within the project «Support for Doctoral Studies at University of Latvia».

References

1. S. Kozlovics, "A Dialog Engine Metamodel for the Transformation-Driven Architecture," in *Scientific Papers, University of Latvia*, vol. 756, pp. 151–170, 2010.
2. K. Freivalds and P. Ķikusts, "Optimum layout adjustment supporting ordering constraints in graph-like diagram drawing," in *Proceedings of the Latvian Academy of Sciences, Section B*, vol. 55, pp. 43–51, 2001.
3. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT Press, Cambridge, MA, U.S.A., 2nd ed., 2001.
4. G. J. Badros, A. Borning, and P. J. Stuckey, "The cassowary linear arithmetic constraint solving algorithm," *ACM Trans. Comput.-Hum. Interact.*, vol. 8, pp. 267–306, Dec. 2001.
5. A. Borning, K. Marriott, P. Stuckey, and Y. Xiao, "Solving linear arithmetic constraints for user interface applications," in *Proceedings of the 10th annual ACM symposium on User interface software and technology*, UIST '97, (New York, NY, USA), pp. 87–96, ACM, 1997.
6. H. Hosobe, "A modular geometric constraint solver for user interface applications," in *Proceedings of the 14th annual ACM symposium on User interface software and technology*, UIST '01, (New York, NY, USA), pp. 91–100, ACM, 2001.
7. "QT Developer Network." `http://qt-project.org/`.
8. "Graphical Modeling Framework (GMF, Eclipse Modeling subproject)." `http://www.eclipse.org/gmf`.

9. A. Shatalin and A. Tikhomirov, "Graphical Modeling Framework architecture overview," in *Eclipse Modeling Symposium*, 2006.
10. S. Cook, G. Jones, S. Kent, and A. Wills, *Domain-Specific Development with Visual Studio DSL Tools*. Addison-Wesley, 2007.
11. A. Kalnins, O. Vilitis, E. Celms, E. Kalnina, A. Sostaks, and J. Barzdins, "Building tools by model transformations in Eclipse," in *Proceedings of DSM'07 Workshop of OOPSLA 2007*, (Montreal, Canada), pp. 194–207, Jyvaskyla University Printing House, 2007.
12. J. Barzdins, G. Barzdins, K. Cerans, R. Liepins, and A. Sprogis, "OWLGrEd: a UML style graphical notation and editor for OWL 2," in *Proceedings of OWLED 2010*, 2010.
13. "The GradeTwo tool." `http://gradetwo.lumii.lv/`.

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2012. Vol. 787
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 125–135 P.

Frequency Computable Relations ⋆

Madara Augstkalne, Anda Beriņa, Rūsiņš Freivalds,

Institute of Mathematics and Computer Science, University of Latvia,
Raiņa Bulvāris 29, Riga, LV-1459, Latvia

Abstract. A transducer is a finite-state automaton with an input and an output. We compare possibilities of nondeterministic and probabilistic transducers, and prove several theorems which establish an infinite hierarchy of relations computed by these transducers. We consider only left-total relations (where for each input value there is exactly one allowed output value) and Las Vegas probabilistic transducers (for which the probability of any false answer is 0). It may seem that such limitations allow determinization of these transducers. Nonetheless, quite the opposite is proved; we show a relation which can only be computed by probabilistic (but not deterministic) transducers, and one that can only be computed by nondeterministic (but not probabilistic) transducers.
Frequency computation was introduced by Rose and McNaughton in early sixties and developed by Trakhtenbrot, Kinber, Degtev, Wechsung, Hinrichs and others. It turns out that for transducers there is an infinite hierarchy of relations computable by frequency transducers and this hierarchy differs very much from similar hierarchies for frequency computation by a) Turing machines, b) polynomial time Turing machines, c) finite state acceptors.

1 Introduction

Frequency computation was introduced by G. Rose [13] as an attempt to have a deterministic mechanism with properties similar to probabilistic algorithms. The definition was as follows. A function f: $w \to w$ is (m,n)-computable, where $1 \leq m \leq n$, iff there exists a recursive function R: $w^n \to w^n$ such that, for all n-tuples $(x_1, \cdots, x_n)$ of distinct natural numbers,

$$card\{i : (R(x_1, \cdots, x_n))_i = f(x_i)\} \geq m.$$

R. McNaughton cites in his survey [12] a problem (posed by J. Myhill) whether f has to be recursive if m is close to n. This problem was answered by B.A. Trakhtenbrot [14] by showing that f is recursive whenever $2m > n$. On the other hand, in [14] it was proved that with $2m = n$ nonrecursive functions can be (m,n)-computed. E.B. Kinber extended the research by considering frequency enumeration of sets [9]. The class of (m,n)-computable sets equals the class of recursive sets if and only if $2m > n$.

⋆ The research was supported by Grant No. 09.1570 from the Latvian Council of Science.

For resource bounded computations, frequency computability behaves differently. For example, it is known that whenever $n' - m' > n - m$, under any reasonable resource bound there are sets which are (m', n')-computable, but not (m, n)-computable. However, for finite automata, an analogue of Trakhtenbrot's result holds: the class of languages (m, n)-recognizable by deterministic finite automata equals the class of regular languages if and only if $2m > n$. Conversely, for $2m \leq n$, the class of languages (m, n)-recognizable by deterministic finite automata is uncountable for a two-letter alphabet [1]. When restricted to a one-letter alphabet, every (m, n)-recognizable language is regular. This was also shown by Kinber.

Frequency computations became increasingly popular when the relation between frequency computation and computation with a small number of queries was discovered [11, 6, 2, 3].

We considered problems similar to those in the classical papers [14, 9, 1] for finite-state transducers. We found the situation to be significantly different.

A finite state transducer is a finite state machine with two tapes: an input tape and an output tape. These tapes are one-way, i.e. the automaton never returns to the symbols once read or written. Transducers compute relations between the input words and output words. A deterministic transducer produces exactly one output word for every input word processed.

In this paper we consider advantages and disadvantages of nondeterministic, deterministic, frequency and probabilistic transducers. Obviously, if a relation is such that several output words are possible for the same input word, then the relation cannot be computed by a deterministic transducer. For this reason, in this paper we restrict ourselves to relations which produce exactly one output word for every input word processed.

Definition 1. *We say that a relation $R(x, y)$ is* left-total*, if for arbitrary x there is exactly one y satisfying $R(x, y)$.*

Probabilistic algorithms may be of several types: those allowing errors of all types, Monte Carlo, Las Vegas, etc. Since our relations produce exactly one output word for every input word processed, it was natural to consider only Las Vegas transducers, i.e. probabilistic transducers for which the probability of every false answer is 0. It follows immediately from the definition that every relation computed by such a probabilistic transducer can be computed by a nondeterministic transducer as well. We prove below that nondeterministic transducers are strictly more powerful than Las Vegas transducers.

The zero probability of all possible false results of a probabilistic transducer has another unexpected consequence. It turns out that only recursive relations of small computational complexity can be computed by probabilistic transducers with a probability, say, $\frac{1}{4}$ or $\frac{1}{7}$. Hence a natural question arises, whether every rational number $0 \leq p \leq 1$ can be the best probability to compute some relation by a probabilistic finite-state transducer. It turns out that it is not the case. We show examples of relations that can be computed by probabilistic finite-state transducers with probabilities $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$, etc. and not better. We believe that no other best probabilities can exist for relations of the type considered by us.

This hierarchy of relations turns out to be related to the number of symbols of help needed for deterministic finite-state transducers that take advice to compute these relations.

2 Definitions

We use standard definitions of deterministic, nondeterministic and probabilistic transducers, which are well-established in theoretical computer science literature [5]. Our model is slightly different in the regard that we allow multiple output symbols for each transition, however it can be easily seen that the expressive power of transducer is unaffected.

Definition 2. *A nondeterministic transducer D is a six-tuple $\langle Q, \Sigma, \Delta, \delta, q_0, F\rangle$, which satisfies the following conditions:*

- *Q is a finite set, whose members are called states of D. $q_0 \in Q$ is called the initial state of D,*
- *Σ is a set of input symbols of D and is called the input alphabet,*
- *Δ is a set of output symbols of D and is called the output alphabet,*
- *δ is a relation from $Q \times (\Sigma \cup \{\epsilon\})$ to $Q \times \Delta^*$. Each tuple $((q, \alpha), (p, \beta))$ is called a transition rule of D,*
- *$F \subseteq Q$ is the set of accepting or final states of D*

A transducer operates in discrete time $t = 1, 2, 3 \ldots$ and handles two one-way tapes, where one is read-only input tape and contains a string from the input alphabet Σ and second is write-only output tape with finite output alphabet Δ.

The computation begins at the start state q_0. The computation from state q proceeds by reading symbol α from the input tape, following a suitable transition rule $((q, \alpha), (p, \beta))$ (the new state being p) and writing the corresponding output β to the output tape. The only exception is so called ϵ-transition $((p, \epsilon), (q, \beta))$, where the transducer changes the state and possibly writes an output without reading an input symbol.

We consider all our transducers as machines working infinetely long. At every moment, let x be the word having been read from the input tape up to this moment, and let y be the word written on the output tape up to this moment. Then we say that the pair (x, y) belongs to the relation computed by the transducer.

Definition 3. *A deterministic transducer is a nondeterministic transducer such that transition relation δ is a function, i.e., for each input state and each input symbol the corresponding output state and output symbol is uniquely determined according to the transition table δ.*

Definition 4. *A probabilistic transducer is a transducer of the form $\langle Q, \Sigma, \Delta, \Psi, q_0, F\rangle$ where the transition function Ψ is probabilistic, i.e., for every state q and input symbol α, output state p and output symbol β there is an*

associated probability with which transition occurs. We furthermore stipulate that probabilities of all transition from any fixed pair of input state and input symbol must sum to one.

Definition 5. *We say that a probabilistic transducer A computes a left-total relation R with probability p if for arbitrary pair $(x, y) \in R$ when A works on input x the probability to go into an accepting state having produced the output y is no less than p.*

Definition 6. *We say that a probabilistic transducer A is a Las Vegas transducer computing a left-total relation R with probability p if for arbitrary pair $(x, y) \in R$ when A works on input x the probability to go into an accepting state having produced the output y is no less than p, and the probability to go into an accepting state having produced the output other than y equals 0.*

Since we consider in our paper only left-total relations and since our probabilistic transducers are Las Vegas, it easily follows that the transducer for arbitrary x and for arbitrary $\alpha \in \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ either produces a correct output y or reaches a non-accepting state.

Definition 7. *A frequency transducer with parameters (m, n) is a transducer with n input tapes and n output tapes. Every state of the transducer is defined as $(a_1, a_2, \cdots, a_n)$-accepting where each $a_i \in \{$ accepting , nonaccepting $\}$. We say that a left-total relation R is (m, n)-computed by the transducer if for arbitrary n-tuple of pairwise distinct input words $(x_1, x_2, \cdots, x_n)$ there exist at least m distinct values of x_i such that the i-th output y_i satisfies $(x_i, y_i) \in R$.*

Please notice that we do not demand the state of the frequency transducer to be all-accepting after the reading of the input words. This allows us introduce a counterpart of Las Vegas transducer.

Definition 8. *A Las Vegas frequency transducer with parameters (m, n) is a transducer with n input tapes and n output tapes. Every state of the transducer is defined as $(a_1, a_2, \cdots, a_n)$-accepting where each $a_i \in \{$ accepting , nonaccepting $\}$. We say that a left-total relation R is (m, n)-computed by the transducer if for arbitrary n-tuple of pairwise distinct input words $(x_1, x_2, \cdots, x_n)$: 1) there exist at least m distinct values of x_i such that the i-th output y_i satisfies $(x_i, y_i) \in R$, and 2) if a result y_i is produced on any output tape i and the current state of the transducer is $(a_1, a_2, \cdots, a_{i-1}, accepting, a_{i+1}, \cdots, a_n)$-accepting, then $R(x_i, y_i)$.*

We consider frequency transducers like all other transducers as machines working infinitely long. At every moment, let x be the word having been read from the input tape up to this moment, and let y be the word written (and accepted) on the output tape up to this moment. Then we say that the pair (x, y) belongs to the relation computed by the transducer. However, in the case when the input alphabet contains less than n letters, there is a problem how to define (m, n)-computation correctly.

Definition 9. *We say that a frequency transducer is performing a* strong (m,n)-*computation if* at moments when all the input words $(x_1, x_2, \cdots, x_n)$ are distinct *there exist at least m distinct values of x_i such that the i-th output y_i satisfies $(x_i, y_i) \in R$.*

Definition 10. *We say that a frequency transducer is performing a* weak (m,n)-*computation with parameters $(b_1, b_2, \cdots b_n)$, where $b_1, b_2, \cdots b_n$ are distinct integers, if the transducer is constructed in such a way that in the beginning of the work the transducer reads the first b_1 symbols from the input word x_1, the first b_2 symbols from the input word b_2, $\cdots$, the first b_n symbols from the input word x_n and at all subsequent moments reads exactly 1 new symbol from every input tape. This ensures that at all moments the input words $(x_1, x_2, \cdots, x_n)$ are distinct. There is no requirement of the correctness of the results when the length of input words is $(b_1, b_2, \cdots b_n)$ but at all moments afterwards there exist at least m distinct values of x_i such that the i-th output y_i satisfies $(x_i, y_i) \in R$.*

3 Frequency transducers

First of all, it should be noted that a frequency transducer does not specify uniquely the relation computed by it. For instance, a transducer with 3 input tapes and 3 output tapes $(2,3)$-computing the relation

$$R(x,y) = \begin{cases} \text{true, if } x = y \text{ and } x \neq 258714, \\ \text{true, if } y = 0 \text{ and } x = 258714, \\ \text{false, if} \qquad\qquad \text{otherwise.} \end{cases}$$

can output $y = x$ for all possible inputs and, nonetheless, the result is always correct for at least 2 out of 3 inputs since the inputs always distinct. Please notice that the program of the frequency transducer does not contain the "magical" number 258714. Hence the number of states for an equivalent deterministic transducer can be enormously larger.

Theorem 1. *There exists a strong finite-state frequency transducer $(1,2)$-computing a continuum of left-total relations.*

Proof. Consider the following frequency transducer with 2 input tapes and 2 output tapes. If the inputs x_1 and x_2 are such that x_1 is an initial fragment of x_2, then the output y_1 equals x_1, and $y_2 = 0$. If the inputs x_1 and x_2 are such that x_2 is an initial fragment of x_1, then the output y_2 equals x_2, and $y_1 = 0$. In all the other cases $y_1 = y_2 = 0$.

Let R be a relation defined by taking an arbitrary infinite sequence ω of zeros and ones, and defining

$$R(x,y) = \begin{cases} \text{true, if } y = x \text{ and } x \text{ is an initial fragment of } \omega, \\ \text{true, if } y = 0 \text{ and } x \text{ is not an initial fragment of } \omega, \\ \text{false, if} \qquad\qquad \text{otherwise.} \end{cases}$$

There is a continuum of such sequences and a continuum of the corresponding relations. Each of them is $(1,2)$-computed by the frequency transducer. □

Theorem 2. *If $2m > n$ and there exists a strong finite-state frequency transducer (m,n)-computing a left-total relation R then R can also be computed by a deterministic finite-state transducer.*

Proof. Let R be a left-total relation and A be a strong finite-state frequency transducer (m,n)-computing it. Let Q be (another) left-total relation (m,n)-computed by the same transducer A. Let $x_1, x_2, \cdots, x_k$ be distinct input words such that the pairs $(x_1, y_1'), (x_2, y_2'), \cdots, (x_k, y_k')$ are in R, the pairs (x_1, y_1''), $(x_2, y_2''), \cdots, (x_k, y_k'')$ are in Q, and $y_1' \neq y_1'', y_2' \neq y_2'', \cdots, y_k' \neq y_k''$. What happens if $k \geq n$ and the transducer gets an input-tuple containing only values from $\{x_1, x_2, \cdots, x_k\}$? The transducer has to output at least m correct values for R and at least m correct values for Q which is impossible because $2m > n$. A careful count shows that $k \leq 2n - 2m$.

Hence for arbitrary given relation R (m,n)-computable by A there is a constant k such that any relation Q (m,n)-computable by A differs from R on no more that k_0 pairs (x,y). Let Q_0 be one of the relations where indeed k such pairs exist, and let $(x_1, y_1'), (x_2, y_2'), \cdots, (x_k, y_k')$ and (x_1, y_1''), $(x_2, y_2''), \cdots, (x_k, y_k'')$ be these pairs. Of course, there is no algorithm to construct k and Q_0 effectively, but they cannot fail to exist. For arbitrary x the value of y such that $(x,y) \in R$ can be calculated effectively using n-tuples of input words involving the input words $x_1, x_2, \cdots, x_k$ and remembering that no relation computed by A can differ from R and from Q_0 in more than k values. Since A is a finite automaton, the standard cut-and-paste arguments show that this computation needs only input words which differ from x only on the last d digits where d is a suitable constant. □

Theorem 3. *There exists a left-total relation R $(2,3)$-computed by a weak finite-state Las Vegas frequency transducer and not computed by any deterministic finite-state transducer.*

Proof. We consider the relation

$$R(x,y) = \begin{cases} \text{true, if } y = 1^{|x|} & \text{and } \mid x \mid \equiv 0 (mod 3), \\ \text{true, if } y = 1^{|x|} & \text{and } \mid x \mid \equiv 1 (mod 3) \\ \text{true, if } y = 1^{|x|-1} & \text{and } \mid x \mid \equiv 2 (mod 3) \\ \text{false, if} & \text{otherwise.} \end{cases}$$

The weak finite-state Las Vegas frequency $(2,3)$-transducer starts with reading 1 symbol from the first input tape, 2 symbols from the second input tape and 3 symbols from the third input tape and reads exactly 1 symbol from each input tape at any moment. Hence at any moment the transducer has no more than one input word x_i with $\mid x_i \mid \equiv 2(mod 3)$. To have a correct result for the other two input words, it suffices to keep the length of the output being equal the length of the corresponding input. In case if the length of the input is $\mid x_i \mid \equiv 2(mod 3)$, the state becomes non accepting.

The relation cannot be computed by a deterministic transducer because the length of the output y decreases when the length of of the input increases from $3k+1$ to $3k+2$. □

This proof can easily be extended to prove the following Theorem 4.

Theorem 4. *If $m < n$ then there exists a left-total relation R (m, n)-computed by a weak finite-state Las Vegas frequency transducer and not computed by any deterministic finite-state transducer.*

Theorem 5. *There exists a left-total relation R $(2, 4)$-computed by a weak finite-state Las Vegas frequency transducer and not computed by any finite-state $(1, 2)$-frequency transducer.*

Proof. We consider the relation

$$R(x,y) = \begin{cases} \text{true, if } y = 1^{|x|} \text{ and } \mid x \mid \equiv 0(mod 9), \\ \text{true, if } y = 1^{|x|} \text{ and } \mid x \mid \equiv 3(mod 9), \\ \text{true, if } y = 1^{|x|} \text{ and } \mid x \mid \equiv 4(mod 9), \\ \text{true, if } y = 1^{|x|} \text{ and } \mid x \mid \equiv 6(mod 9), \\ \text{true, if } y = 1^{|x|} \text{ and } \mid x \mid \equiv 8(mod 9), \\ \text{true, if } y = 0 \quad \text{and } \mid x \mid \equiv 1(mod 9), \\ \text{true, if } y = 0 \quad \text{and } \mid x \mid \equiv 2(mod 9), \\ \text{true, if } y = 0 \quad \text{and } \mid x \mid \equiv 5(mod 9), \\ \text{true, if } y = 0 \quad \text{and } \mid x \mid \equiv 7(mod 9), \\ \text{false, if} \qquad\qquad \text{otherwise.} \end{cases}$$

(1) The weak finite-state Las Vegas frequency $(2, 4)$-transducer starts with reading 1 symbol from the first input tape, 2 symbols from the second input tape and 3 symbols from the third input tape, 4 symbols from fourth input tape and reads exactly 1 symbol from each input tape at any moment. The transducer always outputs $y_i = 1^{|x_i|}$ on the i-th output tape. Since the transducer can count the length of the input modulo 9, the false outputs (in cases $\mid x_i \mid$ congruent to 1,2,5 or 7 (mod 9)) are not accepted and the transducer is Las Vegas.

At every moment the lengths of the input words are $k, k+1, k+2, k+3$ for some natural k. At least two of them are congruent to 0,3,4,6 or 8 (mod 9).

(2) Assume that the relation is $(1, 2)$-computed by a transducer performing a *weak* $(1, 2)$-computation with parameters (b_1, b_2). Whatever the difference $d = b_2 - b_1$, there exists a value of s such that both $s + b_1$ and $s + b_2$ are congruent to 1,2,5 or 7 (mod 9). Hence the transducer produces two wrong results on the pair $1^{s+b_1}, 1^{s+b_2}$ in contradiction with the $(1, 2)$-computability. □

4 Nonconstructive methods for frequency transducers

Unfortunately, it is not clear how to generalize the explicit construction of the relation $R(x, y)$ in Theorem 5 to prove distinction between (m, n)-computability and (km, kn)-computability for weak finite-state frequency transducers. Luckily, there is a non-constructive method to do so. This method is based on usage of algorithmic information theory.

Definition 11. *We define a transformation I which takes words $x \in \{0,1\}^*$ into $I(x) \in \{0,1\}^*$ by the following rule. Every symbol* 0 *is replaced by* 1100110100 *and every symbol* 1 *is replaced by* 1011001010.

Definition 12. *We define a transformation J which takes words $x \in \{0,1\}^*$ into $J(x) \in \{0,1\}^*$ by the following rule. Every symbol* 0 *is replaced by* 0100110100 *and every symbol* 1 *is replaced by* 0011001010.

Lemma 1. *For arbitrary $x \in \{0,1\}^*$ the result of the transformation I is a word $I(x)$ such that $\mid I(x) \mid = 10 \mid x \mid$, and $I(x)$ contains equal number of zeros and ones.*

Lemma 2. *For arbitrary $x \in \{0,1\}^*$ the result of the transformation J is a word $J(x)$ such that $\mid J(x) \mid = 10 \mid x \mid$, and every subword y of $J(x)$ such that $\mid J(x) \mid = 10$ contains no more than 5 symbols* 1.

Definition 13. *We define a transformation K which takes words $x \in \{0,1\}^*$ into a 2-dimensional array*

$$K(x) = \begin{pmatrix} K_{11} & K_{12} & \cdots & K_{1n} \\ K_{21} & K_{22} & \cdots & K_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ K_{n1} & K_{n2} & \cdots & K_{nn} \end{pmatrix}$$

of size $10 \mid x \mid \times\ 10\ \mid x \mid$ *by the following rule. The first row* $\begin{pmatrix} K_{11} & K_{12} & \cdots & K_{1n} \end{pmatrix}$ *copies $I(x)$. Every next row is a cyclic copy of the preceding one:*

$$\begin{pmatrix} K_{s1} & K_{s2} & \cdots & K_{sn} \end{pmatrix} = \begin{pmatrix} K_{(s-1)2} & K_{(s-1)3} & \cdots & K_{(s-1)1} \end{pmatrix}$$

.

Definition 14. *We define a transformation L which takes words $x \in \{0,1\}^*$ into a 2-dimensional array*

$$L(x) = \begin{pmatrix} L_{11} & L_{12} & \cdots & L_{1n} \\ L_{21} & L_{22} & \cdots & L_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ L_{n1} & L_{n2} & \cdots & L_{nn} \end{pmatrix}$$

of size $10 \mid x \mid \times\ 10\ \mid x \mid$ *by the following rule. The first row* $\begin{pmatrix} L_{11} & L_{12} & \cdots & L_{1n} \end{pmatrix}$ *copies $J(x)$. Every next row is a cyclic copy of the preceding one:*

$$\begin{pmatrix} L_{s1} & L_{s2} & \cdots & L_{sn} \end{pmatrix} = \begin{pmatrix} L_{(s-1)2} & L_{(s-1)3} & \cdots & L_{(s-1)1} \end{pmatrix}$$

.

There is a dichotomy: 1) either there exist 4 rows (say, with numbers b_1, b_2, b_3, b_4) in $K(x)$ such that in every column Z such that among the values $K_{(b_1)z}, K_{(b_2)z}, K_{(b_3)z}, K_{(b_4)z}$ there are exactly 2 zeros and 2 ones, or 2) for arbitrary 4 rows (with

numbers b_1, b_2, b_3, b_4 in $K(x)$) there is a column z such that among the values $K_{(b_1)z}, K_{(b_2)z}, K_{(b_3)z}, K_{(b_4)z}$ there are at least 3 values equal to 1. (Please remember that by Lemma 2 the total number of zeros and ones in every row is the same.) We will prove that if the Kolmogorov complexity of x is maximal and the length of x is sufficiently large then the possibility 1) does not exist.

Lemma 3. *If n is sufficiently large and x is a Kolmogorov-maximal word of length n then for arbitrary 4 rows (with numbers b_1, b_2, b_3, b_4 in $K(x)$) there is a column z such that among the values $K_{(b_1)z}, K_{(b_2)z}, K_{(b_3)z}, K_{(b_4)z}$ there are at least 3 values equal to 1.*

Proof. Assume from the contrary that there exist 4 rows (say, with numbers b_1, b_2, b_3, b_4) in $K(x)$ such that in every column z such that among the values $K_{(b_1)z}, K_{(b_2)z}, K_{(b_3)z}, K_{(b_4)z}$ there are exactly 2 zeros and 2 ones. By definition of K, $K_{(b_i)z} = K_{1(z+b_i-1)}$. Hence every assertion "among the values $K_{(b_1)z}, K_{(b_2)z}, K_{(b_3)z}, K_{(b_4)z}$ there are exactly 2 zeros and 2 ones" can be written as "among the values $K_{1(c_1)}, K_{1(c_2)}, K_{1(c_3)}, K_{1(c_4)}$ there are exactly 2 zeros and 2 ones" which is equivalent to "among the values $I(d_1), I(d_2), I(d_3), I(d_4)$ there are exactly 2 zeros and 2 ones".

Every value $I(d)$ was obtained from a single letter in the word x. Namely, the letters $I(10j+1), I(10j+2), \cdots, I(10j+10)$ were obtained from the j-th letter$x(j)$ of x. $I(10j+1) = 1$ both for $x(j)$ being a or b. $I(10j+2) = 1$ if $x(j)$ equals a but not b. $I(10j+3) = 1$ if $x(j)$ equals b but not a. Hence we introduce a functional

$$h(x(j)) = \begin{cases} 1 & , \text{ if } d \equiv 0 (mod 10) \\ \overline{x(j)} & , \text{ if } d \equiv 1 \text{ or } 4 \text{ or } 5 \text{ or } 7 (mod 10) \\ x(j) & , \text{ if } d \equiv 2 \text{ or } 3 \text{ or } 6 \text{ or } 8 (mod 10) \\ 0 & , \text{ if } d \equiv 9 (mod 10) \end{cases}$$

Using this functional we transform every assertion of "among the values $I(d_1), I(d_2), I(d_3), I(d_4)$ there are exactly 2 zeros and 2 ones" type into a Boolean formula "among the values $h(x_{j_1}), h(x_{j_2}), h(x_{j_3}), h(x_{j_4})$ there are exactly 2 zeros and 2 ones".

Let a set S of such Boolean formulas be given. We say that another formula F is independent from the set S if F cannot be proved using formulas from the set S. For instance, if F contains a variable not present in any formula of S then F is independent from S.

Take a large integer n and consider the set T of all binary words from $\{a, b\}^{2n}$. There are 2^{2n} words in T. Let T_1 be the subset of T containing all words with equal number of a's and b's. Cardinality of T_1 equals $2^{2n-o(n)}$. The set S contains $2n$ formulas but not all of them are independent. However, since each formula contains only 4 variables, there are at least $\frac{2n}{4}$ independent formulas in S. Apply one-by-one these independent formulas and removes from T_1 all the words where some formula fails. Notice that application of a new formula independent from the preceding ones remove at least half of the words. Hence after all removals no more than $2^{\frac{3n}{2}-o(n)}$ words remain. Effective enumeration of all the remaining

words and usage of Kolmogorov numbering as in Section 4 gives a method to compress each x to a length not exceeding $\frac{3n}{2} - o(n)$. This contradicts non-compressibility of Kolmogorov-maximal words. □

Since independence of formulas in our argument was based only on the used variables the same argument proves the following lemma.

Lemma 4. *If n is sufficiently large and x is a Kolmogorov-maximal word of length n then for arbitrary 4 rows (with numbers b_1, b_2, b_3, b_4 in $L(x)$) there is a column z such that among the values $L_{(b_1)z}, L_{(b_2)z}, L_{(b_3)z}, L_{(b_4)z}$ there are at least 3 values equal to 1.*

We are ready to prove the main theorem of this paper.

Theorem 6. *There exists a left-total relation R $(3,6)$-computed by a weak finite-state frequency transducer and not computed by any finite-state $(2,4)$-frequency transducer.*

Proof. Consider the relation

$$R(x,y) = \begin{cases} \text{true, if } y = 1^{|x|} \text{ and } \mid x \mid \equiv j(modn) \text{ and } L_{1j} = 0 \\ \text{true, if } y = 0 \quad \text{and } \mid x \mid \equiv j(modn) \text{ and } L_{1j} = 1 \\ \text{false, if} \qquad\qquad \text{otherwise.} \end{cases}$$

where $L(x)$ is as described above. □

References

1. Holger Austinat, Volker Diekert, Ulrich Hertrampf, Holger Petersen. Regular frequency computations. *Theoretical Computer Science,* vol. 330 No. 1, pp. 15–20, 2005.
2. Richard Beigel, William I. Gasarch, Efim B. Kinber. Frequency computation and bounded queries. *Theoretical Computer Science,* vol. 163 No. 1/2, pp. 177–192, 1996.
3. John Case, Susanne Kaufmann, Efim B. Kinber, Martin Kummer. Learning recursive functions from approximations. *Journal of Computer and System Sciences,* vol. 55, No. 1, pp. 183–196, 1997.
4. A.N.Degtev. On (m,n)-computable sets. *Algebraic Systems,* Edited by D.I. Moldavanskij, Ivanovo Gos. Universitet, pp. 88–99, 1981.
5. Eitan Gurari. An Introduction to the Theory of Computation. *Computer Science Press, an imprint of E. H. Freeman,* Chapter 2.2, 1989.
6. Valentina Harizanova, Martin Kummer, Jim Owings. Frequency computations and the cardinality theorem. *The Journal of Symbolic Logic,* vol. 57, No. 2, pp. 682–687, 1992.
7. Maren Hinrichs and Gerd Wechsung. Time bounded frequency computations. *Information and Computation,* vol. 139, pp. 234-257, 1997.
8. Richard M. Karp and Richard Lipton. Turing machines that take advice. *L' Enseignement Mathematique,* vol. 28, pp. 191–209, 1982.

9. Efim B. Kinber. Frequency calculations of general recursive predicates and frequency enumeration of sets. *Soviet Mathematics Doklady,* vol. 13, pp. 873–876, 1972.
10. Efim B. Kinber. Frequency computations in finite automata, *Kibernetika,* No. 2, pp. 7–15, 1976(Russian; English translation in Cybernetics 12 (1976) 179-187).
11. Martin Kummer. A proof of Beigel's Cardinality Conjecture. *The Journal of Symbolic Logic,* vol. 57, No. 2, pp. 677–681, 1992.
12. Robert McNaughton. The Theory of Automata, a Survey. *Advances in Computers,* vol. 2, pp. 379–421, 1961.
13. Gene F. Rose. An extended notion of computability. *Abstracts of International Congress for Logic, Methodology and Philosophy of Science,* p.14, 1960.
14. Boris A. Trakhtenbrot. On the frequency computation of functions. *Algebra i Logika,* vol. 2, pp.25–32, 1964 (Russian)

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2012. Vol. 787
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 136–146 P.

On fault-tolerance of Grover's algorithm

Nikolajs Nahimovs, Alexander Rivosh, Dmitry Kravchenko

Abstract

Grover's algorithm is a quantum search algorithm solving the unstructured search problem of size N in $O(\sqrt{N})$ queries, while any classical algorithm needs $O(N)$ queries [1].

However, if the query transformation might fail (with probability independent of the number of steps of the algorithm), then quantum speed-up disappears: no quantum algorithm can be faster than a classical exhaustive search by more than a constant factor [6].

In this paper we study the effect of a small number of failed queries. We show that k failed queries with a very high probability change the number of actually executed steps of Grover's algorithm from l to $O\left(\frac{l}{\sqrt{k}}\right)$.

1 Introduction

Grover's algorithm is a quantum search algorithm solving the unstructured search problem in about $\frac{\pi}{4}\sqrt{N}$ queries [1]. It has been analysed in great detail. The analysis has been mainly focused on the optimality and generalization of the algorithm [4, 2, 3], as well as on fault-tolerance of the algorithm to various kind of physical errors, such as decoherence or random imperfections in either diffusion transformations or black box queries [8, 7].

In this paper we study fault-tolerance of Grover's algorithm to logical faults, in our case failure of one or more query transformations. Regev and Schiff have shown [6] that if the query transformation fails with a fixed probability (independent of the number of steps of the algorithm), then quantum speed-up disappears: no quantum algorithm can be faster than a classical exhaustive search by more than a constant factor.

We find it interesting to understand what happens if only a small number of failed queries is allowed. We show that even a single failed query can stop the algorithm from finding any of marked elements. Remarkably, this property does not depend on a number of marked elements. This makes the quantum case completely different from the classical case.

A failure of a single or multiple query transformations results in a number of steps not being executed. We show that k failed queries with a very high probability change the number of actually executed steps of Grover's algorithm from l to $O\left(\frac{l}{\sqrt{k}}\right)$.

2 Grover's algorithm

Suppose we have an unstructured search space consisting of N elements $x_1, \ldots, x_N \in \{0, 1\}$. Our task is to find $x_i = 1$ or to conclude that no such x exists.

Grover's algorithm starts with a state $|\psi_{start}\rangle = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} |i\rangle$. Each step of the algorithm consists of two transformations: Q – query transformation defined as $Q|i\rangle = (-1)^{x_i}|i\rangle$ and D – the inversion about average, defined as:

$$D = \begin{bmatrix} -1+\frac{2}{N} & \frac{2}{N} & \cdots & \frac{2}{N} \\ \frac{2}{N} & -1+\frac{2}{N} & \cdots & \frac{2}{N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{2}{N} & \frac{2}{N} & \cdots & -1+\frac{2}{N} \end{bmatrix}.$$

The state of the algorithm after t steps is $|\psi_t\rangle = (DQ)^t|\psi_{start}\rangle$.

Grover's algorithm has been analysed in detail and many facts about it are known. If there is one marked element, the probability of finding it by measuring $|\psi_t\rangle$ reaches $1 - o(1)$ for $t = O(\sqrt{N})$. If there are k marked elements, the probability of finding one of them by measuring $|\psi_t\rangle$ reaches $1 - o(1)$ for $t = O(\sqrt{N/k})$.

3 Grover's algorithm with errors

In this section we study the effect of small number of errors (omitted query transformations) on the transformation sequence of the algorithm. We show that an omission of a single or multiple query transformations is equivalent to replacing a number of steps (DQ transformation pairs) with an identity transformation, that is performing a smaller number of steps.

Let l be a number of steps of the algorithm. We show that $k \ll l$ uniformly distributed independent errors change the transformation sequence of the algorithm from $(DQ)^l$ to $(DQ)^L$, where L is the random variable with expectation $O(\frac{l}{k})$ and standard deviation $O\left(\frac{l}{\sqrt{k}}\right)$. Therefore, with a very high probability the length of the resulting transformation sequence is $O\left(\frac{l}{\sqrt{k}}\right)$ [5].

Our further analysis is focused on the omission of the query transformation Q. However a very similar analysis can be done for an omission of the D transformation.

3.1 Omitting a single query

The transformation sequence of Grover's algorithm is

$$DQ\,DQ\,\ldots\,DQ = (DQ)^l.$$

If we omit a single query transformation the sequence changes to

$$(DQ)^{l_1} D (DQ)^{l_2},$$

where $l_1 + l_2 + 1 = l$, or

$$D(QD)^{l_1}(DQ)^{l_2}.$$

As $DD = QQ = I$ (this follows from the definitions of D and Q transformations), the shortest subsequence will cancel a part of the longest subsequence. More precisely

$$l_1 \geq l_2: \quad D(QD)^{l_1}(DQ)^{l_2} = D(QD)^{l_1 - l_2}$$

$$l_1 < l_2: \quad D(QD)^{l_1}(DQ)^{l_2} = D(DQ)^{l_2 - l_1}.$$

Thus, a single omitted query transformation changes the transformation sequence of the algorithm from $(DQ)^l$ to $(DQ)^{O(|l_1 - l_2|)}$, decreasing the number of successful steps.

Suppose the query transformation can be omitted on a random algorithm step, that is l_1 is a uniformly distributed random variable. The length of the resulting transformation sequence will also be a random variable. Simple calculations show that it has mean $\frac{l}{2} + O(1)$ and variance $\frac{l^2}{12} + O(l)$.

Corollary

A single omitted query transformation on the average will twice decrease the number of successful steps of the algorithm (or will twice increase the average running time of the algorithm).

If the query transformation will be omitted right in the middle of the transformation sequence ($l_1 = l_2$), the number of successful steps will be 0. That is the algorithm will leave the initial state unchanged.

3.2 Omitting multiple queries

The transformation sequence of the algorithm is

$$DQ\,DQ\,\ldots\,DQ = (DQ)^l.$$

If we omit $k-1$ query transformations, the sequence changes to

$$(DQ)^{l_1}D(DQ)^{l_2}D\ldots(DQ)^{l_{k-1}}D(DQ)^{l_k},$$

where $l_1+l_2+\ldots+l_k+(k-1)=l$. By regrouping the brackets we will get

$$(DQ)^{l_1}DD(QD)^{l_2}(DQ)^{l_3}DD(DQ)^{l_4}\ldots =$$

$$(DQ)^{l_1}(QD)^{l_2}(DQ)^{l_3}(DQ)^{l_4}\ldots$$

Transformations Q and D have the following commutativity property:

$$(QD)^i(DQ)^j = (DQ)^j(QD)^i.$$

Thus, the sequence can be rewritten as

$$(DQ)^{l_1+l_3+\ldots}\,(QD)^{l_2+l_4+\ldots}\,.$$

Therefore, k omitted query transformations change the transformation sequence of the algorithm from $(DQ)^l$ to $(DQ)^{O(l_1-l_2+l_3-l_4+\ldots\pm l_k)}$.

We will show that in the continuous approximation case (positions of errors have continuous uniform distributions and $l_1+l_2+\ldots+l_k=l$) the length of the resulting transformation sequence is a random variable with mean 0 (even k) or $\frac{l}{k}$ (odd k) and variance $O\left(\frac{l^2}{k}\right)$. These values perfectly agree with numerical experiment results for discrete case.

Proof of the main result

Suppose we have $k-1$ independent random variables $X_1, X_2, \ldots, X_{k-1}$. Each X_i is uniformly distributed between 0 and l. That is the probability density function of X_i is

$$f_{X_i}(x) = \begin{cases} \frac{1}{l} & x \in [0,l] \\ 0 & x \notin [0,l] \end{cases}$$

and the cumulative distribution function is

$$F_{X_i}(x) = \begin{cases} 0 & x < 0 \\ \frac{x}{l} & x \in [0, l] \\ 1 & x > l \end{cases} .$$

The above random variables split the segment $[0, l]$ into k subsegments $l_1, l_2, \ldots, l_k$. The length of each subsegment is also a random variable.

Let us focus on the subsegment l_1. Probability that $l_1 \leq x$ is the probability that at least one of $X_i \leq x$. Thus, the cumulative distribution function of l_1 is

$$F_{l_1} = 1 - (1 - F_{X_1})(1 - F_{X_2}) \ldots (1 - F_{X_{k-1}})$$

or

$$F_{l_1}(x) = \begin{cases} 0 & x < 0 \\ 1 - (1 - \frac{x}{l})^{k-1} & x \in [0, l] \\ 1 & x > l \end{cases} .$$

The probability density function of l_1 is

$$f_{l_1}(x) = \begin{cases} \frac{k-1}{l}(1 - \frac{x}{l})^{k-2} & x \in [0, l] \\ 0 & x \notin [0, l] \end{cases} .$$

Knowing the probability density function of l_1, we can calculate its mean and variance by using the following formulae:

$$E[X] = \int_{-\infty}^{\infty} x \cdot f_X(x) dx$$

$$E[X^2] = \int_{-\infty}^{\infty} x^2 \cdot f_X(x) dx$$

$$Var[X] = E[X^2] - E[X]^2.$$

We leave out the details of calculation of integrals and give the results.

$$E[l_1] = \int_{-\infty}^{\infty} x \cdot f_{l_1}(x) dx = \int_0^l x \frac{k-1}{l} (1 - \frac{x}{l})^{k-2} dx = \frac{l}{k}$$

$$E[(l_1)^2] = \int_{-\infty}^{\infty} x^2 \cdot f_{l_1}(x)dx = \int_0^l x^2 \frac{k-1}{l}(1-\frac{x}{l})^{k-2}dx = \frac{2l^2}{k(k+1)}$$

$$Var[l_1] = \frac{2l^2}{k(k+1)} - \left(\frac{l}{k}\right)^2 = \frac{k-1}{k+1} \cdot \left(\frac{l}{k}\right)^2.$$

It is easy to see that all l_i subsegments have the same mean and variance. This follows from the fact that all X_i are independent and are uniformly distributed. We should also note that, although X_i are independent random variables, l_i are not independent (the length of one subsegment increases as other decreases and vice versa) .

Now let us focus on $L = l_1 - l_2 + l_3 - ... \pm l_k$. First we will calculate the mean of L. We will use the following well known formulae:

$$E[-X] = -E[X]$$

$$E[X_1 + \ldots + X_k] = E[X_1] + \ldots + E[X_k].$$

As all l_i have the same mean

$$E[L] = E[l_1] - E[l_2] + \ldots \pm E[l_k] = \begin{cases} 0 & k = 2m \\ \\ \frac{l}{k} & k = 2m+1 \end{cases}.$$

Now we will calculate the variance of L. As l_i subsegments are correlated, we have to use the following formula:

$$Var[X_1 + \ldots + X_k] = \sum_{i=1}^{k} Var[X_i] + \sum_{i \neq j} Cov[X_i, X_j]$$

The subsegment covariance can be easily calculated from the following trivial fact:

$$Var(l_1 + \ldots + l_k) = 0.$$

This is so because $l_1 + \ldots + l_k$ is always equal to l. Using the above formula, we will get:

$$Var[l_1 + \ldots + l_k] = \sum_{i=1}^{k} Var[l_i] + \sum_{i \neq j} Cov[l_i, l_j] = 0$$

or

$$\sum_{i=1}^{k} Var[l_i] = -\sum_{i \neq j} Cov[l_i, l_j].$$

As all l_i have the same mean and variance, they will also have the same covariances $Cov[l_i, l_j]$. Using this fact, we will get

$$k \cdot Var[l_i] = -k(k-1) \cdot Cov[l_i, l_j]$$

or

$$Cov[l_i, l_j] = -\frac{1}{k-1} \cdot Var[l_i] = -\frac{1}{k+1} \cdot \left(\frac{l}{k}\right)^2.$$

Now let us return to the variance of L:

$$Var[L] = \sum_{i=1}^{k} Var[l_i] \pm \sum_{i \neq j} Cov[l_i, l_j].$$

Covariance sign will depend on l_i and l_j signs (whether they are the same or not). More precisely:

$$Cov[-X, Y] = Cov[X, -Y] = -Cov[X, Y]$$

$$Cov[-X, -Y] = Cov[X, Y].$$

If $k = 2m$, then m subsegments have plus sign and m subsegments have minus sign. There are $2m(m-1)$ subsegment pairs with the same signs and $2m^2$ subsegment pairs with opposite signs (we should count both (l_i, l_j) and (l_j, l_i) pairs). Thus, we can rewrite the formula as:

$$Var[L] = k \cdot Var[l_i] + Cov[l_i, l_j] \cdot \left(2m(m-1) - 2m^2\right) =$$

$$= k \cdot Var[l_i] - k \cdot Cov[l_i, l_j] =$$

$$= k \cdot Var[l_i] + \frac{k}{k-1} \cdot Var[l_i] =$$

$$= k \cdot Var[l_i] \cdot \frac{k}{k-1}.$$

If $k = 2m+1$, then $m+1$ subsegments have plus sign and m subsegments have minus sign. There are $(m+1)m + m(m-1) = 2m^2$ subsegment pairs with the same signs and $2(m+1)m$ subsegment pairs with opposite signs. Thus, we can rewrite the formula as:

$$Var[L] = k \cdot Var[l_i] + Cov[l_i, l_j] \cdot \left(2m^2 - 2m(m-1)\right) =$$

$$= k \cdot Var[l_i] + (k-1) \cdot Cov[l_i, l_j] =$$
$$= k \cdot Var[l_i] - Var[l_i] =$$
$$= k \cdot Var[l_i] \cdot \frac{k-1}{k}.$$

Using O notation, we can rewrite both cases as $O(k) \cdot Var[l_i] = O\left(\frac{l^2}{k}\right)$. This ends the proof.

Corollary

We have shown that $k-1$ omitted query transformations change the length of the transformation sequence of the algorithm from l to a random variable with mean 0 (even k) or $\frac{l}{k}$ (odd k) and variance $O\left(\frac{l^2}{k}\right)$.

Using Chebyshev's inequality, we can show that with 96% probability L value will lie within five standard deviations from its mean [5]. For large k (but still $k \ll l$) even a more tight bound can be applied. In the next section we will show that the probability distribution of L for large k is close to the normal distribution. Thus, with 99.7% probability L will lie within three standard deviations from the mean.

Therefore, with a very high probability the length of the resulting transformation sequence changes from l to $O\left(\frac{l}{\sqrt{k}}\right)$.

4 Probability distribution of the median

In the previous section we have studied the following model. We have independent random variables $X_1, X_2, \ldots, X_{k-1}$. Each X_i is uniformly distributed between 0 and l. The random variables split the segment $[0, l]$ into k subsegments $l_1, l_2, \ldots, l_k$. Our task was to estimate $L = l_1 - l_2 + l_3 - l_4 + \ldots \pm l_k$. Due to symmetry of l_i, L is equal to $\frac{l}{2} - X_m$, where X_m is the median of $X_1, X_2, \ldots, X_{k-1}$, that is the point separating the higher half of the points from the lower half of the points.

In this section we will show that for a large number of uniformly distributed random variables (points), the probability distribution of the median is close to the normal distribution.

2k + 1 points

Let us consider a real number interval $[-N; N]$ and $2k+1$ random points, each having a uniform distribution. Median is the point number $k+1$.

Probability density of the median at position x, which is at the distance $|x|$ from 0, can be expressed by the formula

$$f_M(x) = \frac{(N-x)^k (N+x)^k}{(2N)^{2k+1}} \times \frac{(2k+1)!}{k!k!} = \frac{\left(N^2-x^2\right)^k (2k)!\,(2k+1)}{(2N)^{2k+1}\,k!k!} \quad (1)$$

Using the Stirling approximation we can rewrite (1):

$$F_M(x) \approx \frac{\left(N^2-x^2\right)^k \sqrt{4\pi k}\left(\frac{2k}{e}\right)^{2k}(2k+1)}{(2N)^{2k+1}\sqrt{2\pi k}\left(\frac{k}{e}\right)^k \sqrt{2\pi k}\left(\frac{k}{e}\right)^k} = \frac{\left(N^2-x^2\right)^k (2k+1)}{2N^{2k+1}\sqrt{\pi k}} \quad (2)$$

$$= \frac{\left(1-\frac{x^2}{N^2}\right)^k (2k+1)}{2N\sqrt{\pi k}} \quad (3)$$

For large k we can approximate $2k+1$ with $2k$:

$$f_M(x) \approx \frac{\left(1-\frac{x^2}{N^2}\right)^k \sqrt{k}}{N\sqrt{\pi}} \quad (4)$$

For small $\frac{x}{N}$ values (4) can be approximated (applying $1-z \approx e^{-z}$) by

$$f_M(x) \approx \frac{\left(e^{-\frac{x^2}{N^2}}\right)^k \sqrt{k}}{N\sqrt{\pi}} = \frac{\sqrt{k}}{N\sqrt{\pi}} e^{-k\frac{x^2}{N^2}} \quad (5)$$

which corresponds to the normal distribution with mean 0 and variance $\frac{N^2}{2k}$.

2k points

Let us consider a real number interval $[-N;N]$ and $2k$ random points, each having a uniform distribution. Median is the point number k.

Probability density of the median at position X, which is at the distance $|X|$ from 0, can be expressed by the formula

$$f_M(x) = \frac{(N-x)^{k-1} (N+x)^k}{(2N)^{2k}} \times \frac{(2k)!}{(k-1)!k!} = \frac{\left(N^2-x^2\right)^k k\,(2k)!}{(2N)^{2k}\,(N-x)k!k!} \quad (6)$$

Using the Stirling approximation we can rewrite (6):

$$f_M(x) \approx \frac{\left(N^2-x^2\right)^k k\sqrt{4\pi k}\left(\frac{2k}{e}\right)^{2k}}{(2N)^{2k}(N-x)\sqrt{2\pi k}\left(\frac{k}{e}\right)^k\sqrt{2\pi k}\left(\frac{k}{e}\right)^k} = \frac{\left(N^2-x^2\right)^k\sqrt{k}}{N^{2k}(N-x)\sqrt{\pi}} \quad (7)$$

$$= \frac{\left(1-\frac{x^2}{N^2}\right)^k\sqrt{k}}{\left(1-\frac{x}{N}\right)N\sqrt{\pi}} \quad (8)$$

For small $\frac{X}{N}$ values (8) can be approximated (applying $1-z \approx e^{-z}$) by

$$f_M(X) \approx \frac{\left(e^{-\frac{x^2}{N^2}}\right)^k\sqrt{k}}{\left(e^{-\frac{x}{N}}\right)N\sqrt{\pi}} = \frac{\sqrt{k}}{N\sqrt{\pi}}e^{-k\frac{x^2}{N^2}+\frac{x}{N}} \quad (9)$$

By multiplying (9) with $e^{-\frac{1}{4k}}$, which for large k is close to 1, we will get

$$f_M(x) \approx \frac{\sqrt{k}}{N\sqrt{\pi}}e^{-k\frac{x^2}{N^2}+\frac{x}{N}-\frac{1}{4k}} = \frac{\sqrt{k}}{N\sqrt{\pi}}e^{-k\frac{(x-\frac{N}{2k})^2}{N^2}} \quad (10)$$

which corresponds to the normal distribution with mean $\frac{N}{2k}$ and variance $\frac{N^2}{2k}$.

5 Conclusions

We have shown that even a single failed query can change the resulting transformation sequence of the algorithm to an identity transformation. On the average, a single failed query will twice decrease the length of the resulting transformation sequence. In case of k failed queries with a very high probability the length of the resulting transformation sequence will be decreased $O(\sqrt{k})$ times.

Similar argument can be applied to a wide range of other quantum query algorithms, such as amplitude amplification, some variants of quantum walks and NAND formula evaluation, etc. That is to any quantum query algorithm for which the transformation X applied between queries has the property $X^2 = I$.

References

[1] Lov Grover
A fast quantum mechanical algorithm for database search.
Proceedings, 28th Annual ACM Symposium on the Theory of Computing (STOC), May 1996, pages 212-219
arXiv:quant-ph/9605043

[2] Cristof Zalka
Grovers quantum searching algorithm is optimal.
Physical Review A, 60:2746-2751, 1999.
arXiv:quant-ph/9711070

[3] G. Brassard et al.
Quantum Amplitude Amplification and Estimation.
arXiv:quant-ph/0005055

[4] C. Bennett et al.
Strengths and Weaknesses of Quantum Computing.
SIAM Journal on Computing (special issue on quantum computing) volume 26, number 5, pages 1510-1523.
arXiv:quant-ph/9701001

[5] R. Motwani, P. Raghavan
Randomized Algorithms.
Cambridge University Press, 1994.

[6] O. Regev, L. Schiff.
Impossibility of a Quantum Speed-up with a Faulty Oracle.
Proceedings of ICALP'2008, Lecture Notes in Computer Science, 5125:773-781, 2008.
`http://www.cs.tau.ac.il/~odedr/papers/faultygrover.pdf`

[7] D. Shapira et al.
Effect of unitary noise on Grover's quantum search algorithm.
Phys. Rev. A volume 67, issue 4, April 2003

[8] Pil H. Song, ILki Kim.
Computational leakage: Grover's algorithm with imperfections.
The European Physical Journal D - Atomic, Molecular, Optical and Plasma Physics volume 23, Number 2, May 2003, pages 299-303.
arXiv:quant-ph/0010075

[9] John Watrous
Quantum Computation.
Lecture course "CPSC 519/619", University of Calgary, 2006.
http://www.cs.uwaterloo.ca/ watrous/lecture-notes.html

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2012. Vol. 787
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 147–152 P.

On languages not recognizable by One-way Measure Many Quantum Finite automaton *

Agnis Škuškovniks

Faculty of Computing, University of Latvia, Raina bulv. 19, Riga, LV-1586, Latvia,
e-mail: agnis.skuskovniks@lu.lv

Abstract. Measure Many Quantum Finite Automaton is not the strongest known model of quantum automaton but, it is still an open question what is the class of languages recognized by it when only bounded error probability is allowed. In this paper a natural class of "word set" regular languages are studied. A proof that in general case the class can not be recognized by One-way model of Measure Many Quantum Finite automata is given.

Keywords: quantum computation, quantum automata, substring, MM-QFA

1 Introduction

Quantum finite automata (QFA) were introduced by several authors in different ways, and they can recognize different classes of languages. The least capable QFA are those (later called MO-QFA) introduced by C.Moore and J.Crutchfield [1]. More powerful notion of QFA (later called MM-QFA) was introduced by A.Kondacs and J.Watrous [2]. They introduced a 1-way and 2-way model of MM-QFA.

A.Ambainis and R. Freivalds[3] examined 1-way model and found that for some languages 1-way MM-QFA (1-QFA) can have exponential size advantages over classical counterparts. They used a language in a single-letter alphabet to prove these advantages. A.Ambainis and other authors[4] improved the base of the exponent in these advantages by using more complicated languages in richer alphabets. Later A.Ambainis together with N.Nahimovs [5] simpified the construction and improved the exponent even more. Nonetheless A.Kondacs and J.Watrous [2] showed that 1-QFA can only recognize regular languages, moreover, 1-QFA cannot recognize all the regular languages.

Future research had emphasis on 2-way automata because it was shown that they could recognize, all regular languages [6]. Also more general 1-way models of QFA where discovered[7][8], that could recognize all regular languages. Later more general models of quantum automata were introduced and studied[9].

There are still open problems for 1-QFA model. There are two models of 1-QFA: one that recognizes languages with bounded error probability and other

* Supported by ESF project 2009/0216/1DP/1.1.1.2.0/09/APIA/VIAA/044.

that recognizes languages with unbounded error probability. It has been shown that unbounded model recognizes all and only stochastic languages[10]. It is not known what is the language class recognized by 1-QFA with bounded error[11]. The study of this class is interesting and challenging because it has been shown that it is not closed under union or any standard Boolean operation[12].

Due to these facts the study of different natural language classes recognizable by 1-QFA is reasonable and justifiable even if stronger models of quantum automata exist. In this paper we study a natural class of "word set" languages. We show that in general case it cannot be recognized by 1-QFA with bounded error.

2 Preliminaries

2.1 1-Way Measure Many Quantum Finite Automata(1-QFA)

We define 1-QFA as follows[13].

Definition 21 *A one-way quantum finite automaton $\mathcal{A}$ is specified by the finite (input) alphabet Σ, the finite set of states Q, the initial state q_0, the sets $Q_a \subseteq Q$ and $Q_r \subseteq Q$ of accepting and rejecting states, respectively, with $Q_a \cap Q_r = empty - set$, and the transition function*

$$\delta : Q \times \Gamma \times Q \to \mathbf{C}_{[0,1]},$$

where $\Gamma = \Sigma \cup \{\#, \$\}$ is the tape alphabet of $\mathcal{A}$, and $\#$ and $\$$ are endmarkers not in Σ. The evolution (computation) of $\mathcal{A}$ is performed in the inner-product space $l_2(Q)$, i.e., with the basis $\{|q\rangle | q \in Q\}$, using the linear operators V_σ, $\sigma \in \Gamma$, defined by

$$V_\sigma(|q\rangle) = \sum_{q' \in Q} \sigma(q, \sigma, q')|q'\rangle,$$

which are required to be unitary.

In computation of $\mathcal{A}$ the so-called computational observable $\mathcal{O}$ is used that corresponds to the orthogonal decomposition

$$l_2(Q) = E_a \oplus E_r \oplus E_n,$$

where $E_a = span\{|q\rangle | q \in Q_a\}$ and $E_r = span\{|q\rangle | q \in Q_r\}$ and E_n is the orthogonal complement of $E_a \oplus E_r$. Denote by P_p,$p \in \{a, r, n\}$ the projection operator into the subspace E_p.
A computation of A on an input $\#\sigma_1 \ldots \sigma_n\$$ proceeds as follows. The operator $V_\#$ is first applied to the starting configuration $|q_0\rangle$ and then the observable $\mathcal{O}$ is applied to the resulting configuration $V_\#|q_0\rangle$. This observable projects $V_\#|q_0\rangle$ into a vector $|\psi'\rangle$ of one of the subspaces E_a, E_r, E_n, with the probability equal to the square of the norm of $|\psi'\rangle$. If $|\psi'\rangle \; in E_a$ the input is accepted; if $|\psi'\rangle \in E_r$the input is rejected. If $|\psi'\rangle \in E_n$, then, after the normalization of $|\psi'\rangle$, the operator $V_{\sigma 1}$ is applied to $|\psi'\rangle$ and after that the observable $\mathcal{O}$ to the resulting vector. This

process goes on. Operators $V_{\sigma 1}, V_{\sigma 2}, \ldots, V_{\sigma n}$ are applied one after another, and after each such application the measurement by the observable $\mathcal{O}$ is performed. In all cases the computation continues only if a projection into E_n occurs.
When no termination occurs the computation can be seen as an application of the composed operator

$$V'_{\sigma n} V'_{\sigma n-1} \ldots V'_{\sigma 1} |q_0\rangle,$$

where $V'_{\sigma 1} = P_n V_{\sigma 1}$.
In order to define formally the overall probability with witch an input is accepted (rejected) by a 1QFA $\mathcal{A}$, we define the set $V_{\mathcal{A}} = l_2(Q) \times \mathbf{C} \times \mathbf{C}$ of so-called "total states" of $\mathcal{A}$, that will be used only with the following interpretation. $\mathcal{A}$ is at any time during the computation in the state (ψ, p_a, p_r) if so far in its computation $\mathcal{A}$ accepted the input with probability p_a, rejected with probability p_r and neither with probability $1 - p_a - p_r = \|\psi\|^2$, and $|\psi\rangle$ is its current, unnormalized state. For each $\sigma \in \Gamma$ the evolution of $\mathcal{A}$, with respect to the total state, on an input σ is given by the operator T_σ defined by

$$T_\sigma(\psi, p_a, p_r) \to (P_n V_\sigma \psi, p_a + \|P_a V_\sigma \psi\|^2, p_r + \|P_r V_\sigma \psi\|^2).$$

For $x = \sigma_1 \sigma_2 \ldots \sigma_n \in \Gamma^*$ let $T_{\#x\$} = T_\$ T_{\sigma n} T_{\sigma n-1} \ldots T_{\sigma 1} T_\#$. If $T_{\#x\$}(|q\rangle, 0, 0) = (\psi, p_a, p_r)$, then we say that $\mathcal{A}$ accepts x with probability p_a and rejects with probability p_r. A 1QFA $\mathcal{A}$ is said to accept a language L with probability $\frac{1}{2} + \epsilon, \epsilon > 0$, if it accepts any $x \in L$ with probability at least $\frac{1}{2} + \epsilon$ and rejects any $x \notin L$ with probability at least $\frac{1}{2} + \epsilon$. If there is an $\epsilon > 0$ such that $\mathcal{A}$ accepts L with probability at least $\frac{1}{2} + \epsilon$,then L is said to be accepted by $\mathcal{A}$ with bounded-error probability. L is said to be accepted with unbounded error probability if $x \in L$ is accepted with probability at least $\frac{1}{2}$ and $x \notin L$ rejected with probability at least $\frac{1}{2}$.
On $V_{\mathcal{A}}$ we define a "norm" $\|\cdot\|_u$ as follows

$$\|(\psi, p_a, p_r)\|_u = \frac{1}{2}(\|\psi\| + |p_a| + |p_r|)$$

and let $D = \{v \in V_{\mathcal{A}} | \|v\|_u \leq 1\}$. D contains all global states of $\mathcal{A}$.

2.2 A useful lemma

Consider the following lemma

Lemma 21 *If $|u\rangle$ and $|v\rangle$ are vectors such that for a linear operator A, reals $0 < \epsilon < 1$ and $\mu > 0$,$\|A(u-v)\| < \epsilon$, and $\|v\|$, $\|u\|$,$\|Au\|$, $\|Av\|$ are in $[\mu, \mu + \epsilon]$, then there is a constant c, that does not depend on ϵ, such that $\|u - v\| < c\epsilon^{\frac{1}{4}}$.*

For proof look in [2]

3 Recognition of "word set" language class

Given a finite alphabet A and a finite set of words in this alphabet $\{w_1, w_2, \ldots, w_k\}$, consider the language class that consists of finite words $\{w_1, w_2, \ldots, w_k\}^*$.

Due to the properties of nondeterministic automata(NFA) it is easy to see that a NFA that recognizes language from this class can easily be constructed. Language that is recognizable by NFA is regular, so languages in "word set" language class are regular as well.

Theorem 31 *Given a language $S = \{w_1, w_2, \ldots, w_k\}^*$ it is not possible in general case to build a 1-QFA that recognizes this language.*

Proof. Lets look at the language $S_1 = \{x | x \in [\{0,1\}^6 1] \cup [\{0,1\}^8 1]\}^*$ and the length of the words are greater than 56 (so that each word length could be put together by parts of length 7 and 9) in other words, language consists of all finite words of length $l = 7 * a + 9 * b$ that end with 1 and other symbols are from $\{0,1\}$.
It is easy to see that this language consists of subset of words from $\{0,1\}^*1$ language. Restrictions for determining the subset are: the word length is grater than 56 and there is no substring that consists of more than 8 following zeros. A.Kondacs and J.Watrous [2] state that:

Theorem 32 *The regular language $L_0 = \{0,1\}^*0$ cannot be recognized by a 1QFA with bounded-error probability.*

Proof. The proof is by contradiction. Let $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, Q_a, Q_r \rangle$ be a 1QFA recognizing the language L_0. To each $x = \sigma_1 \ldots \sigma_n \in \Gamma^*$ we assign the state

$$|\psi_x\rangle = V'_{\sigma_n} \ldots V'_{\sigma_1} |q_0\rangle$$

and let $\mu = inf_{w \in \{0,1\}^*} \{\|\psi_{\#w}\|\}$. For each $w \in \{0,1\}^*$, $w0 \in L_0$ and $w1 \notin L_0$. If $\mu = 0$, then clearly $\mathcal{A}$ cannot recognize L_0 with bounded-error probability $\frac{1}{2} + \epsilon$. Let us therefore assume that $\mu > 0$. For any $\epsilon > 0$ there is a w such that $\|\psi_{\#w}\| < \mu + \epsilon$, and also $\|\psi_{\#wy}\| \in [\mu, \mu + \epsilon]$ for any $y \in \{0,1\}^*$. In particular, for any $m > 0$

$$\|V_1'^m |\psi_{\#w0}\rangle\| \in [\mu, \mu + \epsilon). \tag{1}$$

This implies that the sequence $\{V_1'^i |\psi_{\#w0}\rangle\}_{i=0}^{\infty}$ is bounded in the finite dimensional inner-product space and must have a limit point. Therefore there have to exist j and k such that

$$\|V_1'^j (|\psi_{\#w0}\rangle - V_1'^k |\psi_{\#w0}\rangle)\| < \epsilon.$$

By 21 last inequality together with 1 imply, that there is a constant c which does not depend on ϵ and such that

$$\||\psi_{\#w0}\rangle - V_1'^k |\psi_{\#w0}\rangle\| < c\epsilon^{\frac{1}{4}}.$$

This implies that

$$\|T_{\#w0\$}(|q_0\rangle, 0, 0) - T_{\#01^k\$}(|q_0\rangle, 0, 0)\|_u < c'\epsilon^{\frac{1}{4}}$$

for fixed c'. However, this has to be valid for an arbitrarily small ϵ. This is not possible if $\mathcal{A}$ accepts L_0 because $\mathcal{A}$ should accept the string $w0$ and reject $w01^k$. Hence $\mathcal{A}$ cannot accept L_0 with bounded error probability.
End of proof

From the proof we see that 1-QFA "has problems" recognizing the last 1 of the word, and that other characters are not of an importance. As a result - this theorem also apply to the language S_1. Language S_1 in general case is not recognizable by 1-QFA.QED
End of proof

4 Conclusion

In this paper a One way Measure Many Quantum Finite Automata model was studied. Although there are more general and stronger quantum automata models known, it is still an open question what is the class of languages that this model recognizes.

After studying a "word set" language class a theorem has been proven that there exists a language from "word set" language class that is not recognizable by 1-QFA.

The method of construction of the language is rather interesting, and there is a place for future research to look for specific word sets that can create a 1-QFA recognizable language and perhaps make the size of the minimal 1-QFA recognizing such language exponentially more efficient than deterministic finite automata.

References

1. Moore C., Crutchfield J. *Quantum automata and quantum grammars.* Theoretical Computer Science 237, p.275-306 (2000)
2. Kondacs A., Watrous J. *On the Power of Quantum Finite State Automata.* FOCS, p.66-75 (1998)
3. Ambainis A., Freivalds R. *1-way quantum finite automata: strengths, weaknesses and generalizations.* Proceedings of the 39th Symposium on Foundations of Computer Science,Palo Alto, California, p.332-341 (1998)
4. Ambainis A., Barbans U., Belousova A., Belovs A., Dzelme I., Folkmanis G., Freivalds R., Ledins P., Opmanis R., Skuskovniks A. *Size of Quantum Versus Deterministic Finite Automata.* VLSI, p.303-308 (2003)
5. Ambainis A., Nahimovs N. *Improved constructions of quantum automata.* Theoretical Computer Science 410(20), p.1916-1922 (2009)
6. Freivalds R. *Languages Recognizable by Quantum Finite Automata.* CIAA, p. 1-14 (2005)
7. Ciamarra M.P. *Quantum reversibility and new model of quantum automaton.* FCT v.2138 of LNCS, p.376-379 (2001)
8. Merghetti C., Palano B. *Quantum finite automata with control language. Theoretical Informatics and Applications 40, p.315-322 (2006)*
9. *Ambainis A., Beaudry M., Golovkins M., Kikusts A., Mercer M., Therien D.* Algebraic Results on Quantum Automata. *Theory of Computing Systems 39, 165-188 (2006)*
10. *Yakaryilmaz A., Cem Say A.C.* Languages Recognized with Unbounded Error by Quantum Finite Automata. *CSR, p.356-367 (2009)*
11. *Ambainis A., Kikusts A., Valdats M.* On the Class of Languages Recognizable by 1-way Quantum finite Automata. *STACS, p.75-86 (2001)*
12. *Valdats M.* The class of languages recognizable by 1-way quantum finite automata is not closed under union. *arXiv:quant-ph/0001005v1 (2000)*
13. *Gruska J.* Quantum Computing. *McGraw Hill, p. 153-157 (2000)*

SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2012. Vol. 787
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 153–179 P.

Integer Complexity: Experimental and Analytical Results

Jānis Iraids[1], Kaspars Balodis[2], Juris Čerņenoks[2], Mārtiņš Opmanis[1], Rihards Opmanis[1], and Kārlis Podnieks[1]

[1] Institute of Mathematics and Computer Science, University of Latvia, Raiņa bulvāris 29, Riga, LV-1459, Latvia
[2] University of Latvia, Raiņa bulvāris 19, Riga, LV-1586, Latvia

Abstract. We consider representing of natural numbers by arithmetical expressions using ones, addition, multiplication and parentheses. The (integer) complexity of n – denoted by $\|n\|$ – is defined as the number of ones in the shortest expressions representing n. We arrive here very soon at the problems that are easy to formulate, but (it seems) extremely hard to solve. In this paper we represent our attempts to explore the field by means of experimental mathematics. Having computed the values of $\|n\|$ up to 10^{12} we present our observations. One of them (if true) implies that there is an infinite number of Sophie Germain primes, and even that there is an infinite number of Cunningham chains of length 4 (at least). We prove also some analytical results about integer complexity.

1 Introduction

The field explored in this paper is represented most famously in F26 of Guy [1], and as the sequence A005245 in "The On-Line Encyclopedia of Integer Sequences" [2].

We consider representing of natural numbers by arithmetical expressions using 1's, addition, multiplication and parentheses. Let's call this "representing numbers in basis $\{1, +, \cdot\}$". For example,

$$
\begin{aligned}
&2 = 1 + 1;\\
&3 = 1 + 1 + 1;\\
&4 = 1 + 1 + 1 + 1 = 2 \cdot 2 = (1 + 1) \cdot (1 + 1);\\
&5 = 1 + 1 + 1 + 1 + 1 = 1 + 2 \cdot 2 = 1 + (1 + 1) \cdot (1 + 1);\\
&6 = 1 + 1 + 1 + 1 + 1 + 1 = 2 \cdot 3 = (1 + 1) \cdot (1 + 1 + 1);\\
&7 = 1 + 1 + 1 + 1 + 1 + 1 + 1 = 1 + 2 \cdot 3 = 1 + (1 + 1) \cdot (1 + 1 + 1);\\
&8 = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 =\\
&\quad = 2 + 2 \cdot 3 = 1 + 1 + (1 + 1) \cdot (1 + 1 + 1) =\\
&\quad = 2 \cdot 2 \cdot 2 = (1 + 1) \cdot (1 + 1) \cdot (1 + 1);\\
&9 = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 3 + 2 \cdot 3 = 1 + 2 \cdot 2 \cdot 2 =
\end{aligned}
$$

$$= 3 \cdot 3 = (1+1+1) \cdot (1+1+1);$$
$$10 = 1+1+\ldots+1 = 2 \cdot 2 + 2 \cdot 3 = 2 + 2 \cdot 2 \cdot 2 =$$
$$= 2 \cdot 5 = (1+1) \cdot (1+1+1+1+1);$$
$$11 = \ldots = 2 + 3 \cdot 3 = 1 + 2 \cdot 5;$$
$$12 = \ldots = 2 \cdot 2 \cdot 3;$$
$$13 = \ldots = 2 \cdot 2 + 3 \cdot 3 = 3 + 2 \cdot 5;$$
$$14 = \ldots = 2 \cdot (1 + 2 \cdot 3);$$
$$\ldots$$

As we see, most numbers can be represented in several ways that may differ in size. For example, the number 8 is represented above by three different expressions containing 8, 7 and 6 1's respectively.

We will measure the size of an expression by the number of 1's it contains. We don't need counting neither of operations (if an expression contains k 1's, then it contains k–1 operations), nor of parentheses (the postfix notation might be used).

The size of the shortest expressions representing a particular number n can be considered as the "complexity" of n. Hence, the term "integer complexity". Some numbers allow for several shortest expressions (for examples, see above: 4, 5, 10, 11, 13).

Definition 1. *Let's denote by $\|n\|$ the number of 1's in the shortest expressions representing n in basis $\{1, +, \cdot\}$. We will call it the* ***integer complexity*** *of n.*

For example, as we see above:

$\|1\| = 1;$	$\|5\| = 5;$	$\|9\| = 6;$	$\|13\| = 8;$
$\|2\| = 2;$	$\|6\| = 5;$	$\|10\| = 7;$	$\|14\| = 8;$
$\|3\| = 3;$	$\|7\| = 6;$	$\|11\| = 8;$	$\|15\| = 8;$
$\|4\| = 4;$	$\|8\| = 6;$	$\|12\| = 7;$	$\ldots$

This definition corresponds to the sequence A005245 in "The On-Line Encyclopedia of Integer Sequences". In [1], $\|n\|$ is denoted by $f(n)$, the notation $\|n\|$ is due to Arias de Reyna [3].

In a similar fashion, representation of natural numbers in other bases, for example, $\{1, +, \cdot, -\}$, $\{1, +, \cdot, \uparrow\}$ and $\{1, +, \cdot, -, \uparrow\}$ could also be considered (sequences A091333, A025280 and A091334 [4,5,6], $\uparrow$ stands for exponentiation).

As a function of n, in average, $\|n\|$ is growing logarithmically, namely, one can prove easily:

Theorem 1. *For all $n > 1$,*

$$3 \log_3 n \le \|n\| \le 3 \log_2 n \approx 4.755 \log_3 n$$

In [1], Guy attributes this result to Dan Coppersmith.

The lower bound of Theorem 1 is reached by infinitely many numbers n, exactly – by the powers of three. For example,

$$3^4 = 81 = (1+1+1)\cdot(1+1+1)\cdot(1+1+1)\cdot(1+1+1),$$

and, indeed, products of $1+1+1$s is the best way of representing powers of three in basis $\{1,+,\cdot\}$:

Theorem 2. *$\|n\| = 3\log_3 n$, if and only if $n = 3^b$ for some $b \geq 1$. In particular, for all $b \geq 1$, $\|3^b\| = 3b$ (moreover, the product of $1+1+1$'s is shorter than any other representation of 3^b).*

Similarly, the product of $1+1$s seems to be the best way of representing powers of two in basis $\{1,+,\cdot\}$. For example,

$$2^5 = 32 = (1+1)\cdot(1+1)\cdot(1+1)\cdot(1+1)\cdot(1+1).$$

Hypothesis 1. *For all $a \geq 1$, $\|2^a\| = 2a$ (moreover, the product of $1+1$'s is shorter than any other representation of 2^a).*

$\|2^a\| = 2a$ is true for all powers $2^a < 10^{12}$, i.e. for all a, $0 < a \leq 39$ – as verified by Jānis Iraids. We consider proving or disproving of Hypothesis 1 as one of the biggest challenges of number theory.

The upper bound of Theorem 1 doesn't seem to be exact. As observed by Rawsthorne [7], the "worst" value of $\frac{\|n\|}{\log_3 n}$ seems to be ≈ 3.928 (and not 4.755 of Theorem 1), and it is reached only by a single number, namely, by 1439 ($\|1439\| = 26$):

Hypothesis 2. *For all $n > 1$,*

$$\|n\| \leq \frac{\|1439\|}{\log_3 1439}\log_3 n \approx 3.928\log_3 n$$

Hypothesis 2 is true for all $n \leq 10^{12}$ – as verified by Jānis Iraids.

Thus, we arrive here very soon at the problems that are easy to formulate, but (it seems) extremely hard to solve. In this paper we represent our attempts to explore the field by means of experimental mathematics. We managed to prove analytically only a few of the experimental observations.

In Section 2 we explain the basic concepts and their simplest properties. Section 3 represents our analytical results. Section 4 considers algorithms allowing to calculate the values of $\|n\|$. The best of them was used by Jānis Iraids to calculate $\|n\|$ up to $n = 10^{12}$. In Section 5 we present our experimental observations (several confirmed and refuted hypotheses included).

2 Basic concepts and related work

2.1 The largest and smallest numbers of complexity n

For a given n, there exists only a finite number of expressions of size $\leq n$. Hence,

Definition 2. *a) Let's denote by $E(n)$ the* ***largest*** *m such that $\|m\| = n$.*
b) Let's denote by $E_k(n)$ the ***k-th largest*** *m such that $\|m\| \leq n$ (if it exists). Thus, $E(n) = E_0(n)$.*
c) Let's denote by $e(n)$ the ***smallest*** *m such that $\|m\| = n$.*

In this definition, $E(n)$ corresponds to the sequence A000792[8], and $e(n)$ – to A005520[9].

Proposition 1. *As a function of n, $E(n)$ is monotonically increasing.*

Proof. First, let us note that $E(1) = 1; E(2) = 2; E(3) = 3$. Assume that there is n, such that $E(n+1) < E(n)$. Take the smallest such n. Consider the number $E(n) + 1$. Obviously, $\|E(n) + 1\| \leq n + 1$ and $\|E(n) + 1\| \neq n$. If $\|E(n) + 1\| = k < n$, then $E(k) > E(n)$, that is impossible. Hence, $\|E(n) + 1\| = n + 1$ and $E(n) < E(n + 1)$. □

Proposition 2. *As a function of n, $e(n)$ is monotonically increasing.*

Proof. First, let us note that $e(1) = 1; e(2) = 2; e(3) = 3$. Assume that there is n, such that $e(n) < e(n - 1)$. Take the smallest such n. Consider the number $e(n) - 1$. Obviously, $\|e(n) - 1\| \leq n$, $\|e(n) - 1\| \neq n$ and $\|e(n) - 1\| \neq n - 1$. Hence, $\|e(n) - 1\| \leq n - 2$ and $\|e(n)\| \leq n - 1$, that is impossible. □

Proposition 3. *For all $n \geq 1$, $n \leq E(\|n\|)$.*

Proof. By definition – $E(y)$ is the greatest number whose complexity is y. Thus for all n, if $\|n\| = y$ then $n \leq E(y)$. □

Proposition 4. *For all $x, y \geq 1$, $E(x) \cdot E(y) \leq E(x + y)$.*

Proof. Take the product of shortest expressions for $E(x)$ and $E(y)$. The value of this product is $E(x) \cdot E(y)$ and it contains $x + y$ 1's, so it cannot be greater than $E(x + y)$. □

Theorem 3. *For all $k \geq 0$:*

$$E(3k + 2) = 2 \cdot 3^k;$$

$$E(3k + 3) = 3 \cdot 3^k;$$

$$E(3k + 4) = 4 \cdot 3^k.$$

Guy [10] attributes this result to John L. Selfridge.

Theorem 4. *For all $n \geq 8$:*

$$E_2(n) = \frac{8}{9} E(n).$$

This result is due to Rawsthorne[7].

The behaviour of $e(n)$ appears to be more complicated, for details see Section 5.1.

Lemma 1. *If* $a+b=n$ *and* $\|a\|+\|b\|=\|n\|$ *and* $\|a\|\le\|b\|$ *and given* $\|n\|\le N$, *then* $E(\|a\|)\le\frac{n-\sqrt{n^2-4E(N)}}{2}$.

Proof. From $a+b=n$ we get $E(\|a\|)+E(\|b\|)\ge n$ using Proposition 3. Further, as $\|a\|+\|b\|=\|n\|$, we get $E(\|a\|)+\frac{E(\|n\|)}{E(\|a\|)}\ge n$ by Proposition 4. Still more due to the monotonicity of $E(x)$ we can substitute for the estimate of n's complexity: $E(\|a\|)+\frac{E(N)}{E(\|a\|)}\ge n$. For convenience let us denote $E(\|a\|)$ by x and $E(N)$ by y obtaining $x+\frac{y}{x}\ge n$. Solving the quadratic inequality for x we get $x\le\frac{n-\sqrt{n^2-4y}}{2}$. To complete the proof insert back the original values of x and y. □

Corollary 1. *For* $n\ge 29$, *if* $a+b=n$ *and* $\|a\|+\|b\|=\|n\|$ *and* $\|a\|\le\|b\|$, *then* $a\le 2n^{\log_2 3-1}\approx 2n^{0.585}$.

Proof. By Theorem 1, $N\le 3\log_2 n$. Furthermore, we use the convenient fact that $E(n)\le 3^{\frac{n}{3}}$.

$$\begin{aligned}a\le E(\|a\|)&\le\frac{n-\sqrt{n^2-4E(N)}}{2}\le\\&\le\frac{n-\sqrt{n^2-4\cdot 3^{\log_2 n}}}{2}\le\frac{n\left(1-\sqrt{1-4\cdot n^{\log_2 3-2}}\right)}{2}\le\\&\le\frac{n\left(1-\left(1-4\cdot n^{\log_2 3-2}\right)\right)}{2}\le 2n^{\log_2 3-1}.\end{aligned}$$

□

A similar proof of Corollary 1 is given in [11].

2.2 Ranking numbers

Consider an expression in basis $\{1,+,\cdot\}$ drawn as a rooted n-ary tree, its leaves containing 1's and inner nodes containing either $+$ or $\cdot$. Since both addition and multiplication is associative and commutative, let us merge any adjacent additions and multiplications. For example, the shortest expression for 5^6 can be obtained as follows, it contains 29 1's (not $5\cdot 6=30$, as one might expect):

$$5^6=15625=1+2^3\cdot 3^2\cdot 217=1+2^3\cdot 3^2(1+2^3\cdot 3^3)$$

The corresponding tree is drawn in Figure 1.

Definition 3. *The* ***height of an expression*** *is the height of the corresponding tree.*

In general, for a given n, there can be several shortest expressions of different height, for example, of height 2 and 1:

$$4=(1+1)\cdot(1+1)=1+1+1+1,$$

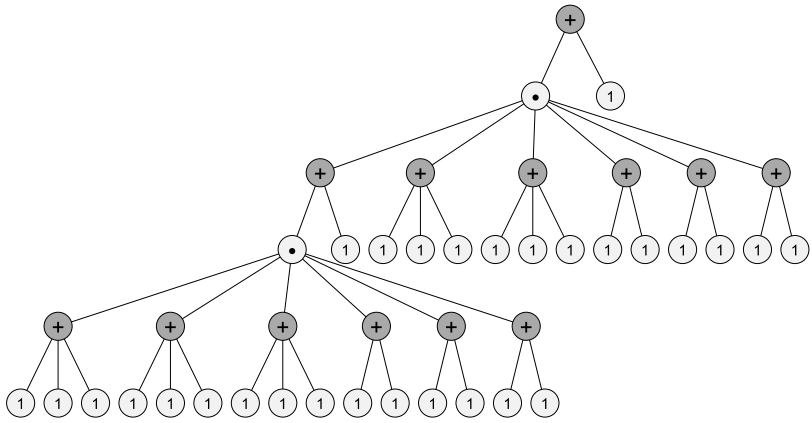


Fig. 1. The tree of the shortest expression for 5^6

or of height 4, 3 and 2:

$$10 = (1+1)\cdot(1+(1+1)\cdot(1+1)) = 1+(1+1+1)\cdot(1+1+1) = (1+1)\cdot(1+1+1+1+1).$$

We will be interested in classifying the shortest expressions by introducing expression height as another complexity measure of positive integers:

Definition 4. *$rank(n)$ – the **rank of a number** n – is defined as the minimum height among the shortest expressions representing n.*

Examples:

- the only number of rank 0 is 1;
- the only numbers of rank 1 are: $2, 3, 4, 5$;
- numbers of rank 2: $6, 8, 9, 10, 12, 15, ...$; see Hypothesis 5 below;
- $rank(7) = rank(11) = rank(13) = 3$;
- $rank(14) = 4$.

For $n \leq 10^9$, $\max rank(n) = 19$ and $\max \|n\| = 67$. For other observations – see Section 5.4.

Obviously $rank(n) = 2$ for infinitely many numbers n. Namely, for all the powers of 3.

There are infinitely many numbers n having $rank(n) = 3$ and 4 as well, see Theorems 6 and 7 below.

Hypothesis 3. *For any $r \geq 2$ there exists an infinite amount of numbers having rank r.*

This hypothesis implies

Hypothesis 4. *As a function of n, $rank(n)$ is unlimited.*

Definition 5. *Let's denote by $r(n)$ the* ***smallest*** *m such that $rank(m) = n$.*

The values of $r(n)$ up to n=19 are represented in Table 4.
Now, let's try exploring more closely the numbers of rank 2.

Lemma 2. *For $n > 1$, the shortest expression of height ≤ 2 representing n, is obtained from the prime decomposition $p_1 \cdot \ldots \cdot p_k$ of n as $(1+\ldots+1)\cdot(1+\ldots+1)\cdot\ldots\cdot(1+\ldots+1)$.*

The number of 1's in this expression, i.e. $p_1 + \ldots + p_k$, is called also the **integer logarithm** of n (see also the sequence A001414[12]).

Theorem 5. *If $rank(n) = 2$, then $n = 2^a3^b5^c$ for some a, b, c with $a+b+c > 1$ and $c < 6$.*

Proof. (Jānis Iraids, Rihards Opmanis) If $n \leq 5$, then $rank(n) \leq 1$. If n is divisible by a prime $p > 5$, then (since $\|p\| < p$) one can convert the expression of Lemma 2 into a shorter expression representing n. Hence, if $rank(n) = 2$, then $n = 2^a3^b5^c$ for some a, b, c with $a + b + c > 1$. But $\left\|5^6\right\| < 5 \cdot 6$ with the shortest expression being of height 5, as seen in Figure 1. Hence, if n is divisible by 5^6, then, again, one can convert the expression of Lemma 2 into a shorter expression representing n. □

Theorem 6. *For any $k > 0$, $rank(3^{4k} + 1) = 3$. Hence, there are infinitely many numbers n having $rank(n) = 3$.*

Proof. (Jānis Iraids) For any $k > 0$, $rank(3^{4k} + 1) \leq 3$. These numbers are not divisible by $3, 4$ and 5, hence, they cannot be represented as $2^a3^b5^c$. According to Theorem 5, this means that $rank(3^{4k} + 1) > 2$. □

Theorem 7. *For any $k > 0$, $rank(3^{4k}(3^{4k} + 1)) = 4$. Hence, there are infinitely many numbers n having $rank(n) = 4$.*

Proof. (Rihards Opmanis) Let us denote $N = 3^{4k}(3^{4k} + 1)$. For any $k > 0$, $rank(N) \leq 4$, and $\|N\| \leq 3 \cdot 8k + 1$.

Since N is not divisible by $3, 4$ and 5, it cannot be represented as $2^a3^b5^c$. According to Theorem 5, this means that $rank(N) > 2$.

It remains to prove that $rank(N) \neq 3$. If $rank(N) = 3$, then the best expression of N has the form $N = X + Y$. We will prove that if $\|X\| \leq \|Y\|$ and $\|X\| + \|Y\| \leq 3 \cdot 8k + 1$, then $X + Y < N$.

Let us recall that

$$E(3n-1) = 2 \cdot 3^{n-1}; E(3n) = 3^n; E(3n+1) = 4 \cdot 3^{n-1}; E(3n+1) = 2 \cdot 3^n.$$

Case 1: $m \geq 1; \|X\| = 3m; 3m \leq \|Y\| \leq 3 \cdot 8k + 1 - 3m = 3(8k - m) + 1$.
Hence, $6m \leq 3 \cdot 8k + 1; m \leq 4k; 8k - m \geq 4k$, and

$$X \leq E(3m) = 3^m; Y \leq E(3(8k-m)+1) = 4 \cdot 3^{8k-m-1}; X+Y \leq 3^m + 4 \cdot 3^{8k-m-1}.$$

Consider the items of the latter sum, their product is constant, hence, the sum will be maximal at $m = 1$ or $m = 4k$, i.e. the maximum will be either

$$3 + 4 \cdot 3^{8k-2} = 3^{8k}(3^{1-8k} + \frac{4}{9}) < 3^{8k}(1 + 3^{-4k}) = N,$$

or

$$3^{4k} + 4 \cdot 3^{4k-1} = 3^{8k}(3^{-4k} + 4 \cdot 3^{-1-4k}) < N.$$

Case 2: $m \geq 0; \|X\| = 3m+1; 3m+1 \leq \|Y\| \leq 3 \cdot 8k+1-3m-1 = 3(8k-m)$. Hence, $6m + 1 \leq 3 \cdot 8k; m \leq 4k - 1; 8k - m \geq 4k + 1$.

If $m = 0$:

$$X = 1; Y \leq E(3 \cdot 8k) = 3^{8k}; X + Y \leq 1 + 3^{8k} < N.$$

If $m > 0$:

$$X \leq E(3m+1) \leq 4 \cdot 3^{m-1}; Y \leq E(3(8k-m)) = 3^{8k-m}; X+Y \leq 4 \cdot 3^{m-1}+3^{8k-m}.$$

Consider the items of the latter sum, their product is constant, hence, the sum will be maximal at $m = 1$ or $m = 4k - 1$, i.e. the maximum will be either

$$4 + 3^{8k-1} = 3^{8k}(4 \cdot 3^{-8k} + \frac{1}{3}) < 3^{8k}(1 + 3^{-4k}) = N,$$

or

$$4 \cdot 3^{4k-2} + 3^{4k+1} = 3^{8k}(4 \cdot 3^{-4k-2} + 3^{1-4k}) < N.$$

Case 3: $m \geq 0; \|X\| = 3m + 2; 3m + 2 \leq \|Y\| \leq 3 \cdot 8k + 1 - 3m - 2 = 3(8k - m) - 1$. Hence, $6m + 3 \leq 3 \cdot 8k; m \leq 4k - 1; 8k - m \geq 4k + 1$.

If $m = 0$:

$$X = 2; Y \leq E(3 \cdot 8k - 1) = 2 \cdot 3^{8k-1}; X + Y \leq 2 + 3^{8k-1} = 3^{8k}(2 \cdot 3^{-8k} + \frac{1}{3}) < N.$$

If $m > 0$:

$$X \leq E(3m+2) \leq 2 \cdot 3^m; Y \leq E(3(8k-m)-1) = 2 \cdot 3^{8k-m-1}; X+Y \leq 2 \cdot 3^m+2 \cdot 3^{8k-m-1}.$$

Consider the items of the latter sum, their product is constant, hence, the sum will be maximal at $m = 1$ or $m = 4k - 1$, i.e. the maximum will be either

$$6 + 2 \cdot 3^{8k-2} = 3^{8k}(6 \cdot 3^{-8k} + \frac{2}{9}) < 3^{8k}(1 + 3^{-4k}) = N,$$

or

$$2 \cdot 3^{4k-1} + 2 \cdot 3^{4k} = 3^{8k}(2 \cdot 3^{-4k-1} + 2 \cdot 3^{-4k}) < N.$$

□

Hypothesis 5. *All and only numbers of rank 2 are the numbers $2^a 3^b 5^c > 5$ with $c < 6$. Or, extending Hypothesis 1:*

$$\left\|2^a 3^b 5^c\right\| = 2a + 3b + 5c$$

for all a, b, c with $a + b + c > 0$ and $c < 6$ (moreover, the product of $1 + ... + 1s$ is shorter than any other representation of $2^a 3^b 5^c$).

As verified by Jānis Iraids, $\left\|2^a 3^b 5^c\right\| = 2a+3b+5c$ is true for all $2^a 3^b 5^c \leq 10^{12}$ with $a + b + c > 0$ and $c < 6$.

For $c = 0$, Hypothesis 5 appears in [7].

2.3 Logarithmic complexity

Because of Theorem 1, the values of $\frac{\|n\|}{\log_3 n}$ are located within the segment $[3, 4.755]$. Hence, the

Definition 6. *The **logarithmic complexity** of $n > 1$ is defined as*

$$\|n\|_{\log} = \frac{\|n\|}{\log_3 n}.$$

For example, $\|2\|_{\log} = \frac{\|2\|}{\log_3 2} \approx 3.1699$.

Hypothesis 6. *It would follow from Hypothesis 5, that the values of $\|n\|_{\log}$ are **dense** across the segment $[3, \|2\|_{\log}]$, i.e. no subsegment of it is free from the values of $\|n\|_{\log}$.*

Proof. According to Hypothesis 5, $\left\|2^a 3^b\right\| = 2a + 3b$, for all $a + b > 0$. Hence, by choosing a and b appropriately, one can locate the value of $\left\|2^a 3^b\right\|_{\log}$ within any given subsegment of $[3, \|2\|_{\log}]$. □

As observed by Mārtiņš Opmanis (and confirmed by Jānis Iraids for all the numbers $\leq 10^{12}$), it seems, the largest values of $\|n\|_{\log}$ are taken by single numbers, see Table 1. The lists in braces represent Cunningham chains of primes [13].

Table 1: Largest values of $\|n\|_{\log}$

n	$\|n\|$	$\approx \|n\|_{\log}$	$rank(n)$	Other properties
1439	26	3.928	9	$e(26), r(9), \{89, 179, 359, 719, 1439, 2879\}$
23	11	3.854	5	$e(11), r(5), \{2, 5, 11, 23, 47\}$
719	23	3.841	7	$e(23)$, {89, 179, 359, 719, 1439, 2879}
179	18	3.812	7	$e(18), r(7)$, {89, 179, 359, 719, 1439, 2879}
4283	29	3.809	7	$e(29), \{2141, 4283\}$
1438	25	3.777	8	$e(25)$, 2×719
59	14	3.772	5	$e(14), \{29, 59\}$
6299	30	3.767	7	$e(30)$, prime
15287	33	3.763	9	$e(33), \{3821, 7643, 15287\}$
107	16	3.762	5	$e(16), \{53, 107\}$
347	20	3.756	7	$e(20), \{173, 347\}$
1499	25	3.756	7	prime
467	21	3.754	5	$e(21), \{233, 467\}$
11807	32	3.749	7	$e(32), \{5903, 11807\}$
263	19	3.746	5	$e(19), \{131, 263\}$
21599	34	3.743	7	$e(34), \{2699, 5399, 10799, 21599\}$

The values of $\|n\|_{\log}$ become (and, it seems, stay) less than 3.60 approximately at $n = 2 \cdot 10^9$.

Let's consider the subsegments $[C, D], C < D$ of the segment $[3, 4.755]$ that do not contain the values of $\|n\|_{\log}$ at all. Of course, according to Hypothesis 6, then $C > \|2\|_{\log}$. Let's denote by C_1 the infimum of these numbers. I.e. C_1 is the point separating the area where the values of $\|n\|_{\log}$ are dense, from the area, where these values are not dense.

Hypothesis 7. *It would follow from Hypothesis 5, that* $C_1 \geq \|2\|_{\log}$.

On the other hand, for some numbers C, $\|n\|_{\log} > C$ only for finitely many values of n. Let's denote by C_2 the infimum of these numbers. This is also known as $\limsup_{n\to\infty} \|n\|_{\log}$. I.e. C_2 is the point separating the area where the values of $\|n\|_{\log}$ are "absolutely sparse", from the area, where these values are not sparse.

Of course, $C_1 \leq C_2$. Hence,

Hypothesis 8. *It would follow from Hypothesis 5, that* $C_2 \geq \|2\|_{\log}$.

More about the possible value of C_2 – in Section 5.1.

3 Analytical results

3.1 Complexity of $2^n - 1$

For the sake of brevity let us introduce $A(n) = \|2^n - 1\| - 2n$ and $B(n) = \|2^n + 1\| - 2n$. We can then establish the following facts.

Theorem 8. *For* $n \geq 1$,

a) $A(2n) \leq A(n) + B(n)$;
b) $A(3n) \leq A(n) + B(n) + 1$;
c) $A(n + 1) \leq A(n) + 1$.

Proof. We shall provide the expressions that will result in these upper bounds.

a) $2^{2n} - 1 = (2^n - 1)(2^n + 1)$. If we take the complexity of both sides and subtract $4n$ we get:

$$A(2n) = \left\|2^{2n} - 1\right\| - 4n \leq \|2^n - 1\| - 2n + \|2^n + 1\| - 2n = A(n) + B(n).$$

b) $2^{3n} - 1 = (2^n - 1)((2^n + 1)2^n + 1)$. Similarly, we get

$$\begin{aligned} A(3n) &= \left\|2^{3n} - 1\right\| - 6n \leq \\ &\leq \|2^n - 1\| - 2n + \|2^n + 1\| - 2n + \|2^n\| - 2n + 1 \leq \\ &\leq A(n) + B(n) + 1. \end{aligned}$$

c) $2^{n+1} - 1 = (2^n - 1) \cdot 2 + 1$. Once again we have

$$A(n+1) = \left\|2^{n+1} - 1\right\| - 2n - 2 \leq \|2^n - 1\| - 2n + 2 - 2 + 1 = A(n) + 1.$$

This method can be extended for numbers other than 2, but then it yields significantly less interesting results because of the very inefficient "$n + 1$" step. □

Corollary 2. *(Kaspars Balodis) If $n > 1$, then*

$$\|2^n - 1\| \leq 2n + \lfloor \log_2 n \rfloor + H(n) - 3,$$

where $H(n)$ is the number of 1's in the binary representation of n, i.e., $H(n)$ is the Hamming weight of n in binary.

Proof. Using the above theorem we can obtain an upper bound of $A(n)$, setting $B(n) = 1$ and $A(1) = A(2) = -1$ and ignoring the rule b) altogether. We will use rule c) per every 1 in the binary representation of n except the leftmost digit, in total, $H(n) - 1$ times. We will use rule a) per every digit, except the two leftmost digits, in total, $\lfloor \log_2 n \rfloor - 1$ times. In this way we will reduce n to 2 at which point $A(n) = -1$ having "paid" 1 for each application of any of the rules a) and c). To sum up:

$$A(n) \leq H(n) - 1 + \lfloor \log_2 n \rfloor - 1 + (-1).$$

□

Corollary 3. *If $n > 1$, then $\|2^n - 1\| \leq 2n + 2 \lfloor \log_2 n \rfloor - 2$.*

3.2 Connection of rank and defect

Definition 7. *(Harry Altman, Joshua Zelinsky, see [14]) The* ***defect of a number*** *n is*

$$d(n) \stackrel{\text{def}}{=} \|n\| - 3 \cdot \log_3 n.$$

Proposition 5. *(Jānis Iraids)*

$$d(n) \geq \left\lfloor \frac{rank(n) - 1}{2} \right\rfloor \left(1 + 3 \log_3 \frac{6}{7}\right).$$

Proof. We will prove this by induction on the rank. First of all, for all n having $rank(n) < 3$ the proposition is true trivially since

$$\left\lfloor \frac{rank(n) - 1}{2} \right\rfloor \left(1 + 3 \log_3 \frac{6}{7}\right) \leq 0.$$

Now assuming that it is true for all n having $rank(n) < r$. Suppose r is an even number, then we again trivially have

$$\left\lfloor \frac{r-1}{2} \right\rfloor \left(1 + 3 \log_3 \frac{6}{7}\right) = \left\lfloor \frac{r-2}{2} \right\rfloor \left(1 + 3 \log_3 \frac{6}{7}\right).$$

If on the other hand, r is an odd number greater than 2, then the shortest expression for n has height at least 3, addition being the outermost operation. Now write n as a sum of numbers of even rank lower than r and order them in non-increasing fashion. This can be done since the numbers of odd rank can be split down further merging them with the n's outermost addition.

$$n = \sum_{i=1}^{k} a_i, \quad rank(a_1) \geq rank(a_2) \geq \cdots \geq rank(a_k).$$

Note that $rank(a_1) = r - 1$ and $k \geq 2$. For defects, we have

$$d(n) = \sum_{i=1}^{k} d(a_i) + 3\log_3 \frac{\prod_{i=1}^{k} a_i}{\sum_{i=1}^{k} a_i}.$$

Following the induction the defect of a_1 is at least $\frac{r-3}{2}\left(1 + 3\log_3 \frac{6}{7}\right)$. If $rank(a_k) \geq 2$ and so all a_i must necessarily be at least 6. The expression $3\log_3 \frac{\prod_{i=1}^{k} a_i}{\sum_{i=1}^{k} a_i}$ is minimised when $k = 2$ and $a_1 = a_2 = 6$ producing a minimum of 3. However, if there are l numbers of rank 0: $rank(a_{k-l+1}) = \ldots = rank(a_k) = 0$ then $d(a_{k-l+1}) = \ldots = d(a_k) = 1$ and $\sum_{i=k-l+1}^{k} d(a_i) + 3\log_3 \frac{\prod_{i=1}^{k} a_i}{\sum_{i=1}^{k} a_i}$ is minimised at $l = 1$, $a_1 = 6$ and $a_2 = 1$ giving the $1 + 3\log_3 \frac{6}{7}$. Consequently,

$$d(n) \geq \frac{r-3}{2}\left(1 + 3\log_3 \frac{6}{7}\right) + \min\left(3, 1 + 3\log_3 \frac{6}{7}\right) =$$

$$= \frac{r-1}{2}\left(1 + 3\log_3 \frac{6}{7}\right).$$

□

4 Algorithms for computing the complexity of a number

The purpose of this chapter is to describe several algorithms for computing $\|n\|$. This is useful for both exploring the behaviour of the complexity and as well as a tool for verifying whether some hypotheses about the value of $\|n\|$ hold for large n.

It appears that computing the complexity of number n is a relatively difficult task. At the time of writing we are unaware of any algorithm that provably works faster than $O(n^{\log_2 3-1})$. Yet there exists an algorithm, that in practice runs in approximately $\Theta(n \log n)$ time.

The **first and simplest algorithm** is essentially as follows: evaluate all the possible expressions of $\{1, +, \cdot\}$ with increasing number of 1's until an expression giving n is found. We can do so taking as basis the postfix notation, also known as Reverse Polish notation. Note, that in the postfix notation the first (leftmost) symbol will always be 1.

The possible expression generator will use three main rules to reduce the number of generated expressions with equal values:

a) After 1 only + or 1 can follow.
b) After + only · or 1 can follow.
c) After · only + or 1 can follow.

For each total number of 1's x in the expression excluding the leading one we can count the number of distinct expressions as 4^x – postfix notation of an expression with $x+1$ 1's contains $2x+1$ symbols and for each symbol there are two possible symbols that can follow. Assuming, that evaluation of the expression takes roughly x time, the overall time complexity can be estimated as $O\left(4^x\right)$. Thus the overall running time of the algorithm will be $O\left(4^{\|n\|}\right)$, which taking into account $\|n\| \leq 3\log_2 n$ will yield $O\left(n^6\right)$.

However, the expression generating algorithm can be improved by eliminating similar expressions of the same number. For instance, 6 can be represented as

* $(1+1)\cdot(1+1+1)$ – postfix notation $11+11+1+\cdot$ or
* $(1+1+1)\cdot(1+1)$ – postfix notation $11+1+11+\cdot$.

Since multiplication is associative, we can omit one of these representations restricting that the first multiplier has to be smaller or equal to the second multiplier. The same principle applies to additions.

The **second algorithm** uses the idea of sieving. We will use an array $f[1..n]$ of integer values where the complexity of the number n will be stored as $f[n]$. Before the main routine starts, precalculate the values of $E(n)$ in an array $E[1..m]$.

for $i = 1$ to n **do**
    $f[i] \leftarrow i$
$updated \leftarrow True$
$height \leftarrow 2$
while $updated$ **do**
    $updated \leftarrow False$
    if $height \equiv 1 \pmod 2$ **then**
        for $i = 2$ to n **do**
            $a \leftarrow \frac{n-\sqrt{n^2-4E[f[n]]}}{2}$
            for $j = 1$ to a **do**
                if $f[i] > f[j] + f[i-j]$ **then**
                    $f[i] \leftarrow f[j] + f[i-j]$
                    $a \leftarrow \frac{n-\sqrt{n^2-4E[f[n]]}}{2}$
                    $updated \leftarrow True$
    else
        for $i = 2$ to $\lfloor\sqrt{n}\rfloor$ **do**
            $j \leftarrow i + i$
            $k \leftarrow 2$
            while $j \leq n$ **do**
                if $f[j] > f[k] + f[i]$ **then**
                    $f[j] \leftarrow f[k] + f[i]$
                    $updated \leftarrow True$
                $j \leftarrow j + i$

$$k \leftarrow k + 1$$
$$height \leftarrow height + 1$$

If the array f is initialized as shown each pass will only update the complexity of numbers that have $rank(n) \geq height$. As a consequence, this algorithm produces the rank of a number; the rank can be stored if necessary. Note that the array f could be initialized with the upper bound provided by Theorem 1 because at no point would $f[i]$ be smaller than $\|i\|$ and yet exceed the value at the corresponding point in unmodified algorithm. To further reduce the running time, one can use a bootstrapping step where numbers that could potentially be used as the smallest of two addends are computed. These are exactly the numbers that cannot be represented best as sums.

While the second algorithm is as fast as any we know, it uses a linear amount of memory – the array where the complexity values are stored. For $n > 10^{11}$ the calculation thus becomes unfeasible. We used this algorithm to calculate $\|n\|$ and $rank(n)$ up to $n = 1.5 \cdot 10^9$.

The **third and final algorithm** – the one we used for calculating $\|n\|$ for n up to 10^{12} – is conceptually very simple: for every natural number compute the complexity by definition and store for subsequent steps.

$$\|1\| = 1 \tag{1}$$
$$\|n\| = \min_{a+b=n \vee a \cdot b = n} \{\|a\| + \|b\|\} \tag{2}$$

The techniques used are identical to what Fuller [15] describes in the comments of his program. For factorisation we used an approach similar to the one described in [16]. The core idea is to maintain a priority queue of the so called eliminators – at any point n for each prime the priority is the smallest integer multiple that is no less than n.

5 Experimental results

5.1 $e(n)$ – the least number of complexity n

Function $e(n)$ corresponds to the sequence A005520 [9]. Our observations up to $n = 89$ are represented in Table 2.

Hypothesis 9. *$e(n)$ is prime for all n, except $n \in \{1, 4, 7, 11, 25\}$.*

Observation 1. *For $k \leq 3$, the number $\frac{e(n)-k}{k+1}$ is prime for almost all n.*

If, for $k = 1$, Observation 1 holds for an infinite number of values of n, it would imply that there is an infinite number of **Sophie Germain primes** – these are defined as integers p such that p and $2p + 1$ are both primes.

Moreover, it seems that the sequence of e(n) contains primes which are the end numbers of increasingly long Cunningham chains. **Cunningham chain** (CC) of length k is defined as a sequence of k primes $\{p_1, p_2, ..., p_k\}$ such that $p_{i+1} = 2 \cdot p_i + 1, 1 \leq i < k$ [13]. In particular,

∗ $e(13)$ is the end number of the first CC of length 5: $\{2, 5, 11, 23, 47\}$;
∗ $e(26)$ is the end number of another CC of length 5: $\{89, 179, 359, 719, 1439\}$;
∗ $e(27)$ is the end number of the first CC of length 6:

$$\{89, 179, 359, 719, 1439, 2879\};$$

∗ $e(80)$ is the end number of another CC of length 5.

The above-mentioned are the only CCs of length ≥ 5 backward-generated by $e(n), n \leq 89$.

For the following 19 values of $n \leq 89$, $e(n)$ generates CCs of length 4:

$$\{11, 23, 34, 49, 51, 60, 61, 65, 66, 67, 70, 72, 73, 74, 77, 84, 86, 87, 89\}.$$

If, for $k = 2; 3$, Observation 1 holds for an infinite number of values of n, it would imply that there is an infinite number of integers p such that p and $3p+2$ (or, correspondingly, p and $4p+3$) are both primes [17].

Table 2: Prime factorizations of numbers close to $e(n)$

n	$\frac{e(n)-2}{3}$	$\frac{e(n)-1}{2}$	$e(n)$	$e(n)+1$
1	–	–	1	2
2	–	–	2	3
3	–	1	3	2^2
4	–	–	2^2	5
5	1	2	5	$2 \cdot 3$
6	5	3	7	2^3
7	–	–	$2 \cdot 5$	11
8	3	5	11	$2^2 \cdot 3$
9	5	2^3	17	$2 \cdot 3^2$
10	–	–	$2 \cdot 11$	23
11	7	11	23	$2^3 \cdot 3$
12	13	$2^2 \cdot 5$	41	$2 \cdot 3 \cdot 7$
13	$3 \cdot 5$	23	47	$2^4 \cdot 3$
14	19	29	59	$2^2 \cdot 3 \cdot 5$
15	29	$2^2 \cdot 11$	89	$2 \cdot 3^2 \cdot 5$
16	$5 \cdot 7$	53	107	$2^2 \cdot 3^3$
17	$5 \cdot 11$	83	167	$2^3 \cdot 3 \cdot 7$
18	59	89	179	$2^2 \cdot 3^2 \cdot 5$
19	$3 \cdot 29$	131	263	$2^3 \cdot 3 \cdot 11$
20	$5 \cdot 23$	173	347	$2^2 \cdot 3 \cdot 29$
21	$5 \cdot 31$	233	467	$2^2 \cdot 3^2 \cdot 13$
22	227	$11 \cdot 31$	683	$2^2 \cdot 3^2 \cdot 19$
23	239	359	719	$2^4 \cdot 3^2 \cdot 5$
24	$11 \cdot 37$	$13 \cdot 47$	1223	$2^3 \cdot 3^2 \cdot 17$
25	–	–	$2 \cdot 719$	1439

26	479	719	1439	$2^5 \cdot 3^2 \cdot 5$
27	$7 \cdot 137$	1439	2879	$2^6 \cdot 3^2 \cdot 5$
28	$5 \cdot 251$	$7 \cdot 269$	3767	$2^3 \cdot 3 \cdot 157$
29	1427	2141	4283	$2^2 \cdot 3^2 \cdot 7 \cdot 17$
30	2099	$47 \cdot 67$	6299	$2^2 \cdot 3^2 \cdot 5^2 \cdot 7$
31	3359	5039	10079	$2^5 \cdot 3^2 \cdot 5 \cdot 7$
32	$5 \cdot 787$	5903	11807	$2^5 \cdot 3^2 \cdot 41$
33	$5 \cdot 1019$	7643	15287	$2^3 \cdot 3 \cdot 7^2 \cdot 13$
34	$23 \cdot 313$	10799	21599	$2^5 \cdot 3^3 \cdot 5^2$
35	$3 \cdot 3733$	$107 \cdot 157$	33599	$2^6 \cdot 3 \cdot 5^2 \cdot 7$
36	$5 \cdot 23 \cdot 131$	$2 \cdot 11299$	45197	$2 \cdot 3^6 \cdot 31$
37	18679	28019	56039	$2^3 \cdot 3 \cdot 5 \cdot 467$
38	$5 \cdot 5443$	40823	81647	$2^4 \cdot 3^6 \cdot 7$
39	32999	49499	98999	$2^3 \cdot 3^2 \cdot 5^3 \cdot 11$
40	54419	81629	163259	$2^2 \cdot 3^2 \cdot 5 \cdot 907$
41	$53 \cdot 1283$	101999	203999	$2^5 \cdot 3 \cdot 5^3 \cdot 17$
42	80627	120941	241883	$2^2 \cdot 3^2 \cdot 6719$
43	$5 \cdot 24763$	185723	371447	$2^3 \cdot 3^2 \cdot 7 \cdot 11 \cdot 67$
44	180179	270269	540539	$2^2 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$
45	196799	295199	590399	$2^6 \cdot 3^2 \cdot 5^2 \cdot 41$
46	302399	453599	907199	$2^6 \cdot 3^4 \cdot 5^2 \cdot 7$
47	$173 \cdot 2083$	540539	1 081 079	$2^3 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$
48	617039	925559	1 851 119	$2^4 \cdot 3^3 \cdot 5 \cdot 857$
49	680399	1 020 599	2 041 199	$2^4 \cdot 3^6 \cdot 5^2 \cdot 7$
50	1 081 079	1 621 619	3 243 239	$2^3 \cdot 3^4 \cdot 5 \cdot 7 \cdot 11 \cdot 13$
51	1 280 159	1 920 239	3 840 479	$2^5 \cdot 3^3 \cdot 5 \cdot 7 \cdot 127$
52	2 187 359	3 281 039	6 562 079	$2^5 \cdot 3^3 \cdot 5 \cdot 7^2 \cdot 31$
53	2 735 519	4 103 279	8 206 559	$2^5 \cdot 3^2 \cdot 5 \cdot 41 \cdot 139$
54	3 898 919	5 848 379	11 696 759	$2^3 \cdot 3^2 \cdot 5 \cdot 32491$
55	4 882 919	7 324 379	14 648 759	$2^3 \cdot 3^2 \cdot 5 \cdot 7 \cdot 5813$
56	$13 \cdot 59 \cdot 9697$	11 156 399	22 312 799	$2^5 \cdot 3^3 \cdot 5^2 \cdot 1033$
57	9 164 959	13 747 439	27 494 879	$2^5 \cdot 3 \cdot 5 \cdot 7^3 \cdot 167$
58	13 915 439	20 873 159	41 746 319	$2^4 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 251$
59	17 417 399	26 126 099	52 252 199	$2^3 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 29$
60	26 110 559	39 165 839	78 331 679	$2^5 \cdot 3^2 \cdot 5 \cdot 7 \cdot 19 \cdot 409$
61	36 202 319	54 303 479	108 606 959	$2^4 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 653$
62	$6577 \cdot 7247$	71 495 279	142 990 559	$2^5 \cdot 3^2 \cdot 5 \cdot 109 \cdot 911$
63	67 699 439	101 549 159	203 098 319	$2^4 \cdot 3^3 \cdot 5 \cdot 17 \cdot 5531$
64	91 328 639	136 992 959	273 985 919	$2^7 \cdot 3^2 \cdot 5 \cdot 13 \cdot 3659$
65	127 340 639	191 010 959	382 021 919	$2^5 \cdot 3^4 \cdot 5 \cdot 7 \cdot 4211$
66	165 145 679	247 718 519	495 437 039	$2^4 \cdot 3^3 \cdot 5 \cdot 7^2 \cdot 31 \cdot 151$
67	227 109 119	340 663 679	681 327 359	$2^8 \cdot 3^2 \cdot 5 \cdot 7^2 \cdot 17 \cdot 71$
68	335 430 119	503 145 179	1 006 290 359	$2^3 \cdot 3^2 \cdot 5 \cdot 601 \cdot 4651$
69	468 798 119	703 197 179	1 406 394 359	$2^3 \cdot 3^3 \cdot 5 \cdot 7 \cdot 17 \cdot 31 \cdot 353$
70	619 264 799	928 897 199	1 857 794 399	$2^5 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 1117$

71	909 474 719	1 364 212 079	2 728 424 159	$2^5 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11^2 \cdot 2237$
72	1 247 732 639	1 871 598 959	3 743 197 919	$2^5 \cdot 3^4 \cdot 5 \cdot 7 \cdot 11^3 \cdot 31$
73	$40499 \cdot 41221$	2 504 113 919	5 008 227 839	$2^9 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 941$
74	2 290 896 719	3 436 345 079	6 872 690 159	$2^4 \cdot 3^3 \cdot 5 \cdot 7 \cdot 454543$
75	3 279 830 399	$7643 \cdot 643693$	9 839 491 199	$2^7 \cdot 3^4 \cdot 5^2 \cdot 7 \cdot 11 \cdot 17 \cdot 29$
76	4 495 159 679	6 742 739 519	13 485 479 039	$2^7 \cdot 3^2 \cdot 5 \cdot 11^3 \cdot 1759$
77	5 574 925 439	8 362 388 159	16 724 776 319	$2^7 \cdot 3^8 \cdot 5 \cdot 7 \cdot 569$
78	8 226 486 239	12 339 729 359	24 679 458 719	$2^5 \cdot 3^2 \cdot 5 \cdot 7 \cdot 19 \cdot 128861$
79	11 841 566 159	17 762 349 239	35 524 698 479	$2^4 \cdot 3^2 \cdot 5 \cdot 49339859$
80	14 737 208 639	22 105 812 959	44 211 625 919	$2^6 \cdot 3^2 \cdot 5 \cdot 7^2 \cdot 11 \cdot 19 \cdot 1499$
81	20 797 230 719	31 195 846 079	62 391 692 159	$2^7 \cdot 3^2 \cdot 5 \cdot 7^2 \cdot 221059$
82	31 251 071 039	46 876 606 559	93 753 213 119	$2^6 \cdot 3^2 \cdot 5 \cdot 7^2 \cdot 61 \cdot 10891$
83	40 517 305 919	60 775 958 879	121 551 917 759	$2^6 \cdot 3^4 \cdot 5 \cdot 7 \cdot 13 \cdot 29 \cdot 1777$
84	54 513 320 399	81 769 980 599	163 539 961 199	$2^4 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 196657$
85	83 528 413 919	125 292 620 879	250 585 241 759	$2^5 \cdot 3^4 \cdot 5 \cdot 7 \cdot 2762183$
86	106 809 776 639	160 214 664 959	320 429 329 919	$2^9 \cdot 3^3 \cdot 5 \cdot 7^2 \cdot 37 \cdot 2557$
87	141 615 840 239	212 423 760 359	424 847 520 719	$2^4 \cdot 3^4 \cdot 5 \cdot 7 \cdot 9366127$
88	210 123 688 319	315 185 532 479	630 371 064 959	$2^7 \cdot 3^3 \cdot 5 \cdot 7 \cdot 13 \cdot 17 \cdot 23581$
89	290 857 880 879	436 286 821 319	872 573 642 639	$2^4 \cdot 3^3 \cdot 5 \cdot 7^3 \cdot 19 \cdot 61987$

The behaviour of $e(n)$ provides some evidence that the logarithmic complexity of n does not tend to 3, and even that the constant $C_2 = \limsup_{n\to\infty} \|n\|_{\log}$ is greater than $\|2\|_{\log}$. It is useful to note the following fact:

Proposition 6.

$$C_2 = \limsup_{n\to\infty} \|n\|_{\log} = \limsup_{n\to\infty} \|e(n)\|_{\log}$$

Proof. Since $\|e(n)\|_{\log}$ is a subsequence of $\|n\|_{\log}$ the $\geq$ follows obviously.

The $\leq$ is proven from the contrary. Assume that there exists an infinite list of numbers $\{x_i\}$, that only have a finite number of elements from sequence $e(n)$ of logarithmic complexity $\geq \min_i\{\|x_i\|_{\log}\}$. Furthermore, there is an infinite subsequence $\{y_i\}$ of $\{x_i\}$'s such that the complexity of numbers strongly increases. But any number of this sequence has a corresponding number in $e(n)$, namely $e(\|y_i\|)$ that has greater or equal logarithmic complexity.

Alternatively, on the axis of logarithmic complexity, $e(n)$ yields the rightmost point of numbers of complexity n. □

Hypothesis 10. *The limit* $\lim_{n\to\infty} \|e(n)\|_{\log}$ *exists and*

$$\lim_{n\to\infty} \|e(n)\|_{\log} = \limsup_{n\to\infty} \|e(n)\|_{\log}$$

As one can see in Figures 2 and 3:

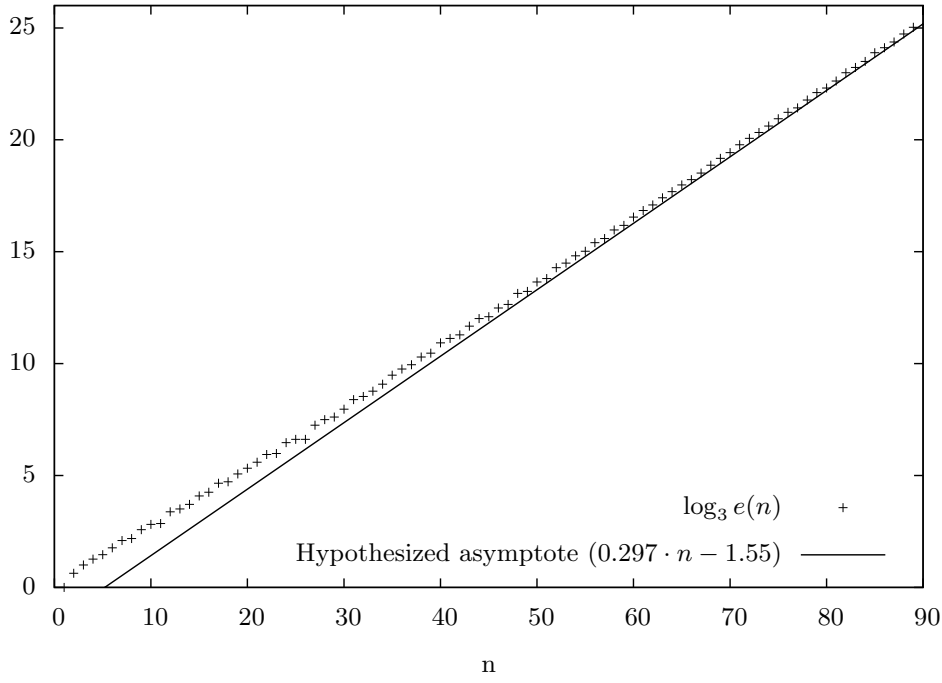


Fig. 2. Values of $\log_3 e(n)$ compared with hypothesized asymptote

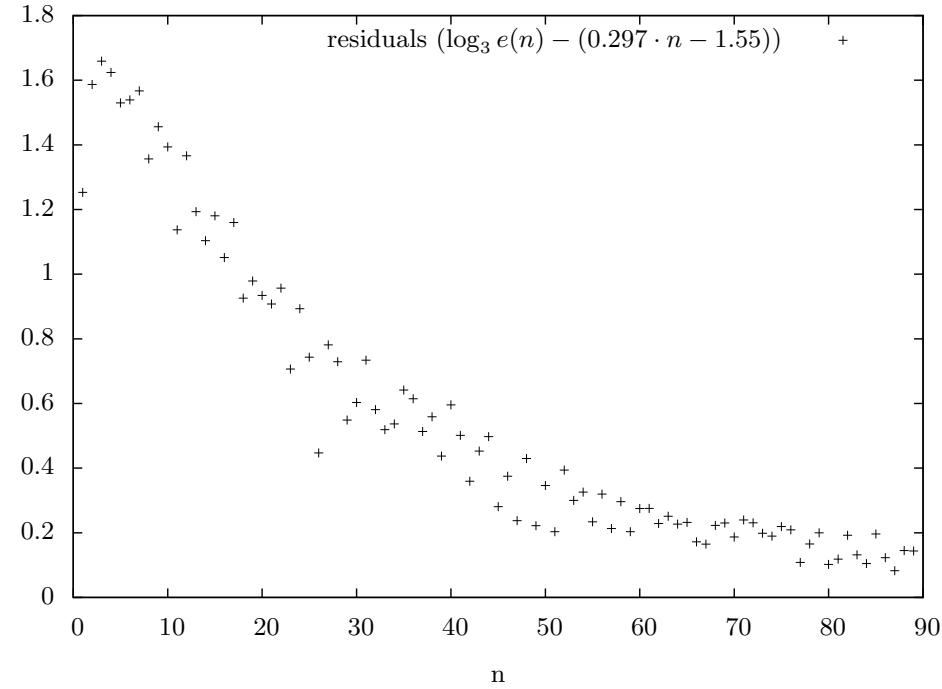


Fig. 3. Residuals of the asymptote

Hypothesis 11. $\log_3 e(n)$ *behaves almost linearly, namely,*

$$\log_3 e(n) \approx 0.297n - 1.55.$$

Hence,

$$\|e(n)\|_{\log} = \frac{n}{\log_3 e(n)} \approx 3.37 + \frac{5.2}{0.297n - 1.55},$$

and $\lim\limits_{n\to\infty} \|e(n)\|_{\log} \approx 3.37$.

Lemma 3. *For all* $n > 1$*:*

$$\|n\|_{\log} \le \|e(\|n\|)\|_{\log}.$$

Proof. Obviously, $\|n\| = \|e(\|n\|)\|$, and $n \ge e(\|n\|)$. Hence, $\log_3 n \ge \log_3 e(\|n\|)$, and $\|n\|_{\log} \le \|e(\|n\|)\|_{\log}$. □

Thus, if $\epsilon > 0$, then, it seems, $\|n\|_{\log} > 3.37 + \epsilon$ can be true only for finitely many values of n. Thus, in terms of Section 2.3:

Hypothesis 12. *It would follow from Hypothesis 11, that* $C_2 \le 3.37$ *(approximately).*

5.2 Structure of shortest expressions

Analyzing the structure of the shortest expressions representing numbers, we have come to the conclusion that we should not only abandon any attempts to obtain a "deterministic" method for construction of shortest expressions, and turn to "nondeterministic" methods. We should abandon also radical attempts to predict regularities in the structure of shortest expressions.

For example, one might propose the following hypothesis: if p is the smallest prime divisor of $n > 1$, then

$$\|n\| = \min\left(1 + \|n-1\|, \|p\| + \|n/p\|\right).$$

For $p = n$, and $p = 2$ (with prime n/p) this hypothesis appears in Guy [10]: for any prime p,

$$\|p\| = 1 + \|p-1\|,$$

$$\|2p\| = \min\left(1 + \|2p-1\|, 2 + \|p\|\right).$$

Similarly, one could suppose:

$$\|3p\| = \min\left(1 + \|3p-1\|, 3 + \|p\|\right).$$

The first hypothesis fails, the smallest counterexample being

$$p = 353\,942\,783 = 2 \cdot 3 + (1 + 2^2 \cdot 3^2)(2 + 3^4(1 + 2 \cdot 3^{10})),$$

$\|p\| = 63, 1 + \|p - 1\| = 64$, found by Martin N. Fuller, 2008, see [2].

The second hypothesis fails as well, the smallest counterexample being

$$2p = 10\,278\,600\,694 = 2 \cdot 3 + (1 + 2 \cdot 3^2(1 + 2^4))(1 + 3^{14}(1 + 2 \cdot 3)),$$

$\|2p\| = 72, 1 + \|2p - 1\| = 2 + \|p\| = 73$, found by Jānis Iraids, 2010.

By analogy, it seems, the third hypothesis also should fail, but this does not happen for $3p \leq 10^{12}$.

Thus, when trying to build the shortest expression representing a number, subtraction of 1 and division by primes are not universal candidates for the first operation. How about subtraction of other numbers?

Subtractions of 3, 4, 5, 7, 10, 11, 13, etc. include subtraction of 1. Thus, it remains to consider only subtractions of 6, 8, 9, 12, etc.

The first number, for which subtraction of 6 is necessary as the first operation, is the above prime found by Fuller:

$$353\,942\,783 = 2 \cdot 3 + (1 + 2^2 \cdot 3^2)(2 + 3^4(1 + 2 \cdot 3^{10})).$$

Until 10^{12}, there are only 21360 numbers for which subtraction of 6, 8 or 9 is necessary as the first operation.

Until 10^{12}, there are exactly 3 numbers for which the first operation must be subtraction of 8:

$$341\,317\,451\,698 = 2 \cdot prime = 2^3 + (1 + 2^4(1 + 2 \cdot 3^3))(1 + 3^{18});$$

$$474\,934\,483\,834 = 2 \cdot 6011 \cdot 39505447 = 2^3 + (1 + 2^4)(1 + 2^3 \cdot 3^4)(1 + 3^{16});$$

$$782\,747\,233\,558 = 2 \cdot prime = 2^3 + (1 + 2^4(1 + 2 \cdot 3^5))(1 + 3^{15}(1 + 2 \cdot 3)).$$

Until 10^{12}, there are 119 numbers for which the first operation must be subtraction of 9, the first three ones being:

$$16\,534\,727\,299 = 103 \cdot 160531333 = 3^2 + (1 + 2^7 \cdot 3^3)(1 + 3^{14});$$

$$68\,238\,632\,999 = prime = 3^2 + (1 + 1 + 3^2)(1 + 2^4 \cdot 3^4)(1 + 3^{14});$$

$$85\,619\,928\,299 = prime = 3^2 + (1 + 2^2 \cdot 3)(1 + 2^4)(1 + 3^{18}).$$

Necessity for subtraction of 12 (or larger addendum) was not detected for numbers until 10^{12}.

According to Corollary 1, if $n \geq 29$ and the shortest expression for n is a sum $n = a + b$, then the smaller addendum $a \leq 2n^{\log_2 3 - 1} \approx 2n^{0.585}$. However, the above observations show that for $n \leq 10^{12}$ the smaller addendum does not exceed 9.

5.3 Complexity of $2^n + 1$ and $2^n - 1$

Since, it seems, $\|2^n\| = 2n$, one can suppose

Hypothesis 13. *For all $n \geq 0$, except 3 and 9,*

$$\|2^n + 1\| = 2n + 1.$$

Hypothesis 13 is true for all $2^n + 1 \leq 10^{12}$, i.e. for all $n \leq 39$ – as verified by Jānis Iraids.

Both exceptions are due to relatively massive divisibility by 3:

$$2^3 + 1 = 9 = 2 \cdot 2 \cdot 2 + 1 = 3 \cdot 3;$$

$$2^9 + 1 = 513 = (3 \cdot 3 \cdot 2 + 1) \cdot 3 \cdot 3 \cdot 3.$$

On the other hand, since we do not have subtraction in our expression basis $\{1, +, \cdot\}$, the numbers $2^n - 1$ seem to be more complicated than 2^n. In Theorem 2, an upper bound of $\|2^n - 1\|$ was proved. In Table 3 this result is compared with experimental data for $n \leq 39$.

Table 3: Complexity of $\|2^n - 1\|$

n	$\|2^n - 1\| - 2n$	$\lfloor \log_2 n \rfloor + H(n) - 3$	n	$\|2^n - 1\| - 2n$	$\lfloor \log_2 n \rfloor + H(n) - 3$
1	−1	−	21	2	4
2	−1	−1	22	3	4
3	0	0	23	4	5
4	0	0	24	2	3
5	1	1	25	3	4
6	0	1	26	3	4
7	1	2	27	2	5
8	1	1	28	3	4
9	1	2	29	4	5
10	2	2	30	3	5
11	3	3	31	4	6
12	1	2	32	3	3
13	2	3	33	4	4
14	2	3	34	4	4
15	2	4	35	4	5
16	2	2	36	2	4
17	3	3	37	2	5
18	1	3	38	2	5
19	2	4	39	3	6
20	2	3	−	−	−

Thus, it seems, the upper bound of Theorem 2 is exact for all $n = 2^k, k > 0$:

Observation 2. *For all* $0 \leq k \leq 5$,

$$\left\| 2^{2^k} - 1 \right\| = 2 \cdot 2^k + k - 2.$$

5.4 Observing ranks

The first number for which subtraction of 6 is necessary as the first operation to obtain the shortest expression of minimum height, is

$$22\,697\,747 = prime = 2 \cdot 3 + (2 + 3^7)(1 + 2^7 \cdot 3^4),$$

complexity 55, rank 5.

The values of $r(n)$ up to $n = 19$ are represented in Table 4. The lists in braces represent Cunningham chains of primes [13].

Table 4: $r(n)$ – the least number of rank n

n	$r(n)$	$\|r(n)\|$	Other properties
1	2	2	$e(2), \{2, 5, 11, 23, 47\}$
2	6	5	$2 \cdot 3$
3	7	6	$e(6), \{3, 7\}$
4	14	8	$2 \cdot 7$
5	23	11	$e(11), \{2, 5, 11, 23, 47\}$
6	86	14	$2 \cdot 43$
7	179	18	$e(18), \{89, 179, 359, 719, 1439, 2879\}$
8	538	21	$2 \cdot 269$
9	1439	26	$e(26), \{89, 179, 359, 719, 1439, 2879\}$
10	9566	30	$2 \cdot 4783$
11	21383	33	$\{10691, 21383, 42767\}$
12	122847	37	$3 \cdot 40949$
13	777419	44	prime
14	1965374	46	$2 \cdot 982687$
15	6803099	51	$\{3401549, 6803099\}$
16	19860614	53	$2 \cdot 9930307$
17	26489579	55	$\{13244789, 26489579, 52979159\}$
18	269998838	61	$2 \cdot 4093 \cdot 32983$
19	477028439	64	$14207 \cdot 33577$

In Figure 4, the values of $\log_3 r(n)$ are compared with n.

Observation 3. $\log_3 r(n)$ *tends to* n*, hence, it seems,* $r(n) \approx 3^n$.

5.5 Collapse of powers

While attempting to prove or disprove the Hypothesis 1 one might try to generalize the hypothesis by looking for other numbers n, such that the shortest

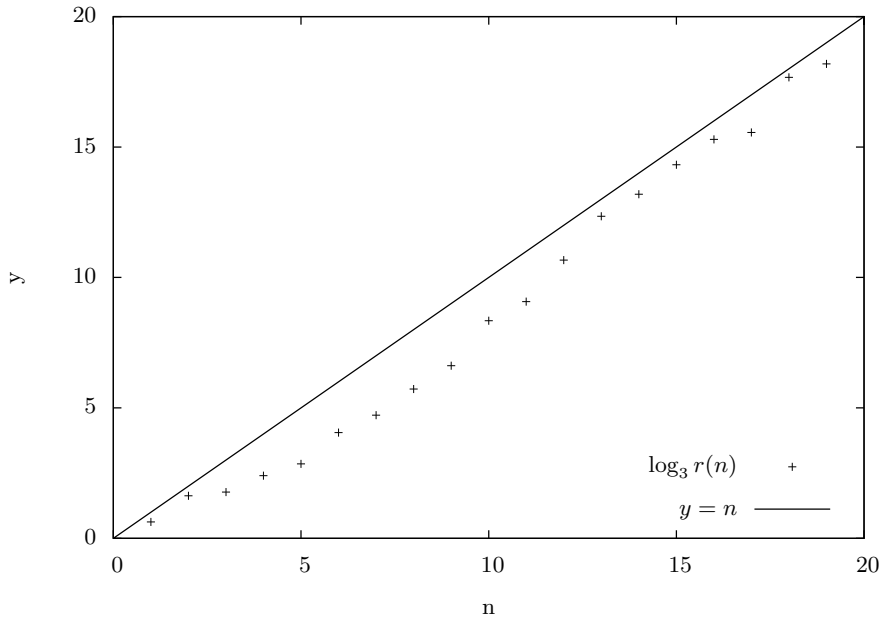


Fig. 4. Values of $\log_3 r(n)$ compared with n.

expressions for all the integer powers of n: n^2, n^3, n^4, ... can be obtained as products of shortest expressions for n.

Definition 8. *If this property holds, then the number n is called* ***resistant****.*

Obviously, if n is resistant, then for all $k > 0$,

$$\left\|n^k\right\| = k \left\|n\right\|;$$
$$\left\|n^k\right\|_{\log} = \left\|n\right\|_{\log}.$$

Currently, only the number 3 is proved to be resistant. According to Hypothesis 1, another resistance candidate is the number 2. Existence of resistant numbers other than powers of 3 would provide a lower bound on C_2:

Proposition 7. *If $n = 2$ or $n \neq 3^k$ is resistant, $C_2 \geq \left\|n\right\|_{\log} > 3$.*

Definition 9. *If $k > 0$ is the least number such that $\left\|n^k\right\| < k \left\|n\right\|$, let us say that the number n* ***collapses at*** *k.*

For example, the number 5 collapses at 6 (see Section 2.2):

$$\left\|5^6\right\| = 29 < 6 \left\|5\right\| = 6 \cdot 5 = 30.$$

However, it seems, most primes collapse already at 2, for example,

$$\left\|11^2\right\| = 15 < 2\,\|11\| = 2 \cdot 8 = 16; 11^2 = 121 = 1 + 2^3 \cdot 3 \cdot 5.$$

Of the 168 primes until 10^3, 120 primes collapse at 2 (71%), 24 – at 3 (14%), 3 – at 4 (2%), the remaining 21 do not collapse until 4 (13%), and, currently, 7 of them are not known to collapse at all, namely,

$$2, 3, 109, 163, 433, 487, 577.$$

More detailed data is represented in Table 5. For numbers whose collapsing power does not exceed 10^{12} the exact value is given. For some of the other numbers, Juris Čerņenoks and Jānis Iraids have used specific methods and specific software to obtain expressions representing powers n^k and containing less than $k\,\|n\|$ 1's. These numbers n are labeled $\leq k$. For example,

$$733^6 = 155104303499468569 =$$

$$2 \cdot 3^2 \cdot (2^5 \cdot 3^2 \cdot (2 \cdot 3 + 1)(2^2 \cdot 3 + 1)(2 \cdot 3^7 \cdot (2 \cdot 3 + 1)(2 \cdot 3^3 + 1) + 1) + 1) \cdot$$
$$(2 \cdot 3 + 1)(2 \cdot 3^2 + 1)(3^2 \cdot (2 \cdot 3^4 + 1) + 1) + 1,$$

where the expression contains $119 < 120 = 6 \cdot \|733\|$ 1's.

The remaining numbers all collapse at 2 or 3, and their logarithmic complexity exceeds 3.417. It seems, primes having larger logarithmic complexity are more likely to collapse.

Table 5: Powers in which primes until 1000 collapse, sorted by increasing logarithmic complexity

p	$collapses$	$\|p\|$	$rank(p)$	$\approx \|p\|_{\log}$
3	–	3	1	3.000
2	$?(> 39)$	2	1	3.170
487	$?(> 4)$	18	3	3.196
163	$?(> 5)$	15	3	3.235
433	$?(> 4)$	18	3	3.257
109	$?(> 5)$	14	3	3.278
811	≤ 9	20	3	3.280
577	$?(> 4)$	19	3	3.283
769	3	20	3	3.307
757	≤ 6	20	5	3.314
541	≤ 6	19	3	3.317
739	≤ 6	20	5	3.326
73	6	13	3	3.329
379	≤ 6	18	5	3.331
733	≤ 6	20	5	3.331
271	4	17	3	3.334
193	4	16	3	3.340

991	≤ 12	21	5	3.344
37	5	11	3	3.347
977	2	21	5	3.351
19	6	9	3	3.358
97	6	14	3	3.362
257	≤ 6	17	3	3.366
937	3	21	5	3.372
673	3	20	5	3.374
919	3	21	5	3.381
181	3	16	3	3.381
661	3	20	5	3.384
7	9	6	3	3.387
653	2	20	5	3.390
337	3	18	5	3.398
641	4	20	3	3.400
883	3	21	5	3.401
127	2	15	5	3.402
881	2	21	5	3.402
877	3	21	5	3.405
241	3	17	3	3.405
631	2	20	5	3.408
457	3	19	5	3.408
331	3	18	5	3.408
5	6	5	1	3.413

Of the 1229 primes until 10^4, 1030 primes collapse at 2 (84%), 122 – at 3 (10%), the remaining 77 do not collapse until 3 (6%).

Of the 78498 primes until 10^6, 71391 primes collapse at 2 (91%), the remaining 7107 do not collapse until 2 (9%), and, currently, only 2722 are not known to collapse at all (4%).

The following observations are true for primes less than 10^6.

Observation 4. *Almost all primes collapse at* 2*.*

Observation 5. *If a prime p does not collapse at all, then $\|p\|_{\log} < 3.364$. If a prime p collapses, then $\|p\|_{\log} > 3.180 > \|2\|_{\log}$.*

If we turn to composite numbers, we can encounter numbers that collapse because one ore more of their prime divisors collapse – even when their initial shortest expression did not directly contain the prime as a multiplier. $3^4+1 = 82$ is an example of such a number:

* $\|82\| = 13$ and $\|41\| = 12$
* $\|82^{12}\| < 12 \cdot 13$, because $\|41^{12}\| \leq 131 < 12 \cdot 11$:

$$\begin{aligned} 41^{12} = &[2^3 \cdot 3^2 \cdot (2^2+1)\,(3^7+1) \\ &(2 \cdot 3^4 \cdot (2^3 \cdot 3^2 \cdot (2 \cdot 3^3+1)+1)+1) \\ &(2 \cdot 3^4 \cdot (2 \cdot 3^9 \cdot (2 \cdot 3+1)+1)+1)] + 1 \end{aligned}$$

On the other hand, there could possibly be composite numbers that do not collapse even though some of their prime divisors do. Obviously, a necessary condition is that the shortest expression for the composite number does not directly contain the collapsing prime as a multiplier. One candidate is $3^5 + 1 = 244 = 2^2 \cdot 61$; $\|244\| = 16$ but $\|61\| = 13$. Using a heuristic algorithm Juris Čerņenoks was able to produce expressions for 61^k for $k = 7..15$. His results suggest that 61 does not collapse well enough, i.e., the number of 1's saved from collapsing is less than required to catch up with the expression $244 = 3^5 + 1$.

On a side note, the above mentioned 82 and 244 are interesting because if they proved to be resistant, then C_2 would exceed $\|2\|_{\log}$.

6 Conclusions

Trying to explore representing of natural numbers by arithmetical expressions using 1's, addition, multiplication and parentheses, one arrives very soon at the problems that are easy to formulate, but (it seems) extremely hard to solve.

Consider, for example, the above Hypothesis 1 stating that the best way of representing of 2^n is $(1 + 1)(1 + 1)$.... We consider proving or disproving of Hypothesis 1 as one of the biggest challenges of number theory.

Almost as challenging seems Hypothesis 4 stating that as a function of n, $rank(n)$ is unlimited. Rank is an additional (to $\|n\|$) measure of integer complexity introduced in this paper.

As another challenge we regard determining of the distribution of the values of logarithmic complexity $\|n\|_{\log}$ which are located within the segment $[3, 4.755]$. First, denote by C_1 the point separating the area (on the left) where the values of $\|n\|_{\log}$ are dense, from the area, where these values are not dense. On the other hand, on the right, the values of $\|n\|_{\log}$ are "absolutely sparse": for most C, $\|n\|_{\log} > C$ only for finitely many values of n. Denote by C_2 the point separating the area (on the right) where the values of $\|n\|_{\log}$ are "absolutely sparse", from the area, where these values are not sparse. Of course, $C_1 \leq C_2$. Our Hypotheses 5 and 11 imply that

$$3.1699 \approx \|2\|_{\log} \leq C_1 \leq C_2 \leq 3.37.$$

Our main experimental "device" is the database containing the values of $\|n\|$ up to $n = 10^{12}$ calculated by Jānis Iraids. The database can be accessed by using an online calculator page linked from [2].

And finally, our Hypothesis 9 and Observation 1 (if true) imply that there is an infinite number of Sophie Germain primes, and even that there is an infinite number of Cunningham chains of length 4 (at least).

References

1. Guy, R.K.: F26 Expressing numbers with just ones. In: Unsolved Problems in Number Theory. Springer Science+Business Media, Inc. (2004)
2. Sloane, N.J.A.: The On-Line Encyclopedia of Integer Sequences. A005245 Complexity of n: number of 1's required to build n using $+$ and $\cdot$.
3. Arias de Reyna, J.: Complejidad de los números naturales. Gaceta de la Real Sociedad Matemática Española (2000)
4. Voss, J.: The On-Line Encyclopedia of Integer Sequences. A091333 Number of 1's required to build n using $+$, $-$, $\cdot$ and parentheses.
5. N. J. A. Sloane, David W. Wilson: The On-Line Encyclopedia of Integer Sequences. A025280 Complexity of n: number of 1's required to build n using $+$, $\cdot$ and $\uparrow$.
6. Voss, J.: The On-Line Encyclopedia of Integer Sequences. A091334 Number of 1's required to build n using $+$, $-$, $\cdot$, $\uparrow$ and parentheses.
7. Rawsthorne, D.A.: How many 1's are needed? Fibonacci Quarterly (1989)
8. Sloane, N.J.A.: The On-Line Encyclopedia of Integer Sequences. A000792 $a(n) = max(n-i)a(i) : i < n; a(0) = 1$.
9. Sloane, N.J.A.: The On-Line Encyclopedia of Integer Sequences. A005520 Smallest number of complexity n: smallest number requiring n 1's to build using $+$ and $\cdot$.
10. Guy, R.K.: Some suspiciously simple sequences. The American Mathematical Monthly (1986)
11. Srinivas Vivek V., Shankar B. R.: Integer complexity: Breaking the $\theta(n^2)$ barrier. World Academy of Science, Engineering and Technology (2008)
12. Sloane, N.J.A.: The On-Line Encyclopedia of Integer Sequences. A001414 Integer log of n: sum of primes dividing n (with repetition).
13. Caldwell, C.K.: Cunningham chain
14. Harry Altman, Joshua Zelinsky: Numbers with Integer Complexity Close to the Lower Bound. `http://arxiv.org/abs/1207.4841` [Last accessed 15 September 2012] (2012)
15. Fuller, M.N.: Program to calculate A005245, A005520, A005421. `http://oeis.org/A005245/a005245.c.txt` [Last accessed: 30 December 2011]
16. Bennet, T.W.: Prime Generator Algorithm. `http://sandbox.mc.edu/~bennet/cs220/codeex/pgenintro.html` [Last accessed 13 January 2012]
17. Caldwell, C.K.: Dicksons conjecture. `http://primes.utm.edu/glossary/xpage/DicksonsConjecture.html` [Last accessed 13 March 2012]

LATVIJAS UNIVERSITĀTES RAKSTI
787. sējums. Datorzinātne un informācijas tehnoloģijas. 2012

Latvijas Universitātes Akadēmiskais apgāds
Baznīcas iela 5, Rīga, LV-1010
Tālr. 67034535