PSI Conference Series

PSI 19

12th A. P. Ershov Informatics Conference

Preliminary Proceedings

July 2–5, 2019
Akademgorodok, Novosibirsk
Russia

A.P. Ershov Institute of Informatics Systems SB RAS
Ministry of Science and Higher Education of the Russian Federation
Novosibirsk State University

A. P. Ershov Informatics Conference

PSI Conference Series, 12th Edition

July 2–5, 2019, Novosibirsk, Akademgorodok, Russia

Preliminary Proceedings

N. Bjørner, I. Virbitskaite, A. Voronkov, Eds.

Organizers

A. P. Ershov Institute of Informatics Systems SB RAS
Novosibirsk State University

Sponsors

RFBR
Microsoft Research
Novosibirsk State University
STI International
STI Innsbruck
ARQA Technologies
Sibers

Novosibirsk
2019

UDK 519.6

A. P. Ershov Informatics Conference : Preliminary Proceedings / A. P. Ershov Institute of Informatics Systems. – Novosibirsk: IPC NSU, 2019. – 366 p.

ISBN 978-5-4437-0911-6

This volume comprises the papers chosen for presentation at A.P. Ershov Informatics Conference (PSI Series, 12th Edition) to be held in Akademgorodok, Novosibirsk, Russia on July 2–5, 2019. The main goal of the conference is to give an overview of research directions in computer science.

UDK 519.6

RFBR Grant № 19-07-20041

ISBN 978-5-4437-0911-6

Conference Chair

Alexander Marchuk
A.P. Ershov Institute of Informatics Systems &
Novosibirsk State University
Novosibirsk, Russia

Steering Committee

Kim Guldstrand Larsen
Aalborg University, Denmark
Bertrand Meyer
ETH, Zurich, Switzerland & Innopolis University,
Kazan, Russia
Vladimiro Sassone
University of Southampton, Great Britain
Michael Gerard Hinchey
Limerick, Ireland
Sriram Rajamani
Microsoft Research India

Program Committee Chairs

Nikolaj Bjørner
Microsoft Research, Redmond, USA
Irina Virbitskaite
A.P. Ershov Institute of Informatics Systems & Novosibirsk State University, Novosibirsk, Russia
Andrei Voronkov
University of Manchester, Great Britain

Keynote Speakers

Moshe Vardi
Rice University, USA
Sören Auer
Leibniz Information Centre for Science and Technology and University Library, Germany
Joost-Pieter Katoen
Aachen University, Germany
Marta Kwiatkowska
Oxford University, UK
Margus Veanes
Microsoft Research, Redmond, USA

Program Committee Members

Farhad Arbab, CWI and Leiden University, The Netherlands
David Aspinall, Edinburgh University, Scotland
Marcello M. Bersani, Politecnico di Milano, Italy
Leopoldo Bertossi, Carleton University, Canada
Andrea Calì, London University, UK
Marsha Chechik, Toronto University, Canada
Volker Diekert, Stuttgart University, Germany
Salvatore Distefano, Messina University, Italy
Nicola Dragoni, Technical University of Denmark, Denmark
Schahram Dustdar, TU Wien, Austria
Dieter Fensel, STI Innsbruck, Austria
Carlo A. Furia, Università della Svizzera Italiana, Italy
Sergei Gorlatch, Muenster University, Germany
Arie Gurfinkel, Carnegie Mellon University, USA
Konstantin Korovin, Manchester University, UK
Maciej Koutny, Newcastle University, UK
Laura Kovács, Vienna University of Technology, Austria
Manuel Mazzara, Innopolis University, Russia
Klaus Meer, Brandenburg University of Technology Cottbus–Senftenberg, Germany
Torben Ægidius Mogensen, DIKU, Denmark
Peter D. Mosses, Swansea University, UK
José Ramón Paramá Gabía, University of A Coruña, Spain
Wojciech Penczek, Institute of Computer Science PAS, Poland
Alexander Petrenko, Institute for System Programming, Russia
Qiang Qu, SIAT, China
Wolfgang Reisig, Humboldt-University of Berlin, Germany
Andrei Sabelfeld, Chalmers University of Technology, Sweden
Davide Sangiorgi, Bologna University, Italy
Cristina Seceleanu, Mälardalen University, Sweden
Natalia Sidorova, Technical University Eindhoven, The Netherlands
Giancarlo Succi, Innopolis University, Russia
Mark Trachtenbrot, Holon Institute of Technology, Israel

Additional Reviewers

Angele, Kevin
Beecks, Christian
Blanck, Jens
De Masellis, Riccardo
Di Iorio, Angelo
Enoiu, Eduard Paul
Fey, Florian
Huaman, Elwin
Jaroszewicz, Szymon
Khazeev, Mansur
Knapik, Michał
Kumar, Vivek
Kunnappilly, Ashalatha
Kuznetsov, Sergei O.
Mahmud, Nesredin
Panasiuk, Oleksandra
Robillard, Simon
Silva-Coira, Fernando
Sourdis, Ioannis
Spina, Cinzia Incoronata
Teisseyre, Pawe
Tomak, Juri
Zubair, Adam

Author' Index

Preface

PSI is the premier international forum in Russia for academic and industrial researchers, developers, and users working on topics relating to computer, software and information sciences. The conference serves to bridge the gaps between different communities whose research areas are covered by but not limited to foundations of program and system development and analysis, programming methodology and software engineering, and information technologies.

The previous eleven PSI conferences were held in 1991, 1996, 1999, 2001, 2003, 2006, 2009, 2011, 2014, 2015, and 2017 and proved to be significant international events. Traditionally, the PSI offers a program of keynote lectures, presentations of contributed papers and workshops complemented by a social program reflecting the amazing diversity of Russian culture and history.

The PSI conference series is dedicated to the memory of a pioneer in theoretical and system programming research, Academician Andrei Petrovich Ershov (1931–1988). Andrei Ershov graduated from Moscow State University in 1954. He began his scientific career under the guidance of Professor Lyapunov, supervisor of his PhD thesis. A.P. Ershov worked at the Institute of Precise Mechanics and Computing Machinery; later, he headed the Theoretical Programming Department at the Computing Center of the USSR Academy of Sciences in Moscow. In 1958, the Department was reorganized into the Institute of Mathematics of the Siberian Branch of the USSR Academy of Sciences, and at the initiative of the Academician S.L. Sobolev, Andrei Ershov was appointed as head of this department, which later became part of the Computing Center in Novosibirsk Akademgorodok. The first important project of the Department was the development of the ALPHA system, an optimizing compiler for an Algol 60 extension implemented on the Soviet computer M-20. Later, the researchers of the Department created the Algibr, Epsilon, Sigma, and Alpha-6 programming systems for the BESM-6 computers. Their achievements include the first Soviet time-sharing system AIST-0, multi-language system BETA, research projects in artificial intelligence and parallel programming, integrated tools for text processing and publishing, and many others. A.P. Ershov led these projects and participated in them. In 1974, he was nominated as a Distinguished Fellow of the British Computer Society. In 1981, he received the Silver Core Award for the services rendered to IFIP. Andrei Ershov's brilliant speeches were always in the focus of public attention. Especially notable was his lecture "The Aesthetic and Human Factor in Programming" presented at the AFIPS Spring Joint Computer Conference in 1972.

This edition of the conference has attracted 70 submissions from 15 countries. We wish to thank all the authors for their interest in PSI 2019. Each submission was reviewed by three experts, with at least two of them being from the same or closely related discipline as the authors. The reviewers generally provided high quality assessment of the papers and often gave extensive comments to the authors so that the contributions could be improved. As a result, the Program Committee has selected 9 high-quality papers as regular talks, 9 papers as short talks, 3 papers as system and experimental talks, and 8 posters for presentation at the conference. Five keynote talks given by prominent computer scientists from various countries cover a range of hot topics in computer science and informatics.

We are glad to express our gratitude to all the people and organizations who have contributed to the conference: the authors of all the papers for their effort in producing the materials included here; the sponsors for their moral, financial and organizational support; the Steering Committee members for their coordination of the conference; the Program Committee members and reviewers for doing their best to review and select the papers, and the Organizing Committee members for their contribution to the success of this event and its great cultural program.

The Program Committee work was done using the EasyChair conference management system.

June, 2019

Nikolaj Bjørner
Irina Virbitskaite
Andrei Voronkov

Contents

Towards Knowledge Graph Based Representation, Augmentation and Exploration of Scholarly Communication

Sören Auer

Leibniz Information Centre for Science and Technology and University Library, Germany

Despite an improved digital access to scientific publications in the last decades, the fundamental principles of scholarly communication remain unchanged and continue to be largely document-based. The document-oriented workflows in science have reached the limits of adequacy as highlighted by recent discussions on the increasing proliferation of scientific literature, the deficiency of peer-review and the reproducibility crisis. We need to represent, analyse, augment and exploit scholarly communication in a knowledge-based way by expressing and linking scientific contributions and related artefacts through semantically rich, interlinked knowledge graphs. This should be based on deep semantic representation of scientific contributions, their manual, crowd-sourced and automatic augmentation and finally the intuitive exploration and interaction employing question answering on the resulting scientific knowledge base. We need to synergistically combine automated extraction and augmentation techniques, with large-scale collaboration to reach an unprecedented level of knowledge graph breadth and depth. As a result, knowledge-based information flows can facilitate completely new ways of search and exploration. The efficiency and effectiveness of scholarly communication will significant increase, since ambiguities are reduced, reproducibility is facilitated, redundancy is avoided, provenance and contributions can be better traced and the interconnections of research contributions are made more explicit and transparent. In this talk we will present first steps in this direction in the context of our Open Research Knowledge Graph initiative and the ScienceGRAPH project.

On Termination of Probabilistic Programs

Joost-Pieter Katoen

Aachen University, Germany

Program termination is a key question in program verification. This talk considers the termination of probabilistic programs, programs that can describe randomised algorithms and more recently received attention in machine learning. Termination of probabilistic programs has some unexpected effects. Such programs may diverge with zero probability; they almost-surely terminate (AST). Running two AST-programs in sequence that both have a finite expected termination time -- so-called positive AST -- may yield an AST-program with an infinite termination time (in expectation). Thus positive AST is not compositional with respect to sequential program composition. This talk discusses that proving positive AST (and AST) is harder than the halting problem, shows a powerful proof rule for deciding AST, and sketches a Dijkstra-like weakest precondition calculus for proving positive AST in a fully compositional manner.

Safety verification for deep neural networks with provable guarantees

Marta Kwiatkowska

University of Oxford, USA

Deep neural networks have achieved impressive experimental results in image classification, but can surprisingly be unstable with respect to adversarial perturbations, that is, minimal changes to the input image that cause the network to misclassify it. With potential applications including perception modules and end-to-end controllers for self-driving cars, this raises concerns about their safety. This lecture will describe progress with developing automated verification and testing techniques for deep neural networks to ensure safety and security of their classification decisions with respect to input manipulations. The techniques exploit Lipschitz continuity of the networks and aim to approximate, for a given set of inputs, the reachable set of network outputs in terms of lower and upper bounds, in anytime manner, with provable guarantees. We develop novel algorithms based on feature-guided search, games and global optimisation, and evaluate them on state-of-the-art networks. We also develop foundations for probabilistic safety verification for Gaussian processes, with application to neural networks.

The lecture will be based on the following publications:

1. X. Huang, M. Kwiatkowska, S. Wang and M. Wu, Safety Verification of Deep Neural Networks. In *Proc. 29th International Conference on Computer Aided Verification (CAV),* pages 3-29, LNCS, Springer, 2017.
2. W. Ruan, X. Huang, and M. Kwiatkowska. Reachability Analysis of Deep Neural Networks with Provable Guarantees. In *Proc. 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*, pages 2651-2659, 2018.
3. M. Wicker, X. Huang, and M. Kwiatkowska. Feature-Guided Black-Box Safety Testing of Deep Neural Networks. In *Proc. 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2018)*, pages 408-426. Springer, 2018.
4. M. Wu, M. Wicker, W. Ruan, X. Huang and M. Kwiatkowska. A Game-Based Approximate Verification of Deep Neural Networks with Provable Guarantees. Accepted to *Theoretical Computer Science* subject to revisions. CoRR abs/1807.03571 (2018)
5. L. Cardelli, M. Kwiatkowska, L. Laurenti, A. Patane. Robustness Guarantees for Bayesian Inference with Gaussian Processes. In *Proc. AAAI 2019.* To appear, 2019. CoRR abs/1809.06452 (2018)

Automated-Reasoning Revolution: From Theory to Practice and Back

Moshe Vardi

Rice University, USA

For the past 40 years computer scientists generally believed that NP-complete problems are intractable. In particular, Boolean satisfiability (SAT), as a paradigmatic automated-reasoning problem, has been considered to be intractable. Over the past 20 years, however, there has been a quiet, but dramatic, revolution, and very large SAT instances are now being solved routinely as part of software and hardware design. In this talk I will review this amazing development and show how automated reasoning is now an industrial reality.

I will then describe how we can leverage SAT solving to accomplish other automated-reasoning tasks. Sampling uniformly at random satisfying truth assignments of a given Boolean formula or counting the number of such assignments are both fundamental computational problems in computer science with applications in software testing, software synthesis, machine learning, personalized learning, and more. While the theory of these problems has been thoroughly investigated since the 1980s, approximation algorithms developed by theoreticians do not scale up to industrial-sized instances. Algorithms used by the industry offer better scalability, but give up certain correctness guarantees to achieve scalability. We describe a novel approach, based on universal hashing and Satisfiability Modulo Theory, that scales to formulas with hundreds of thousands of variables without giving up correctness guarantees.

The Power of Symbolic Automata and Transducers

Margus Veanes

Microsoft Research, Redmond, USA

Symbolic automata and transducers extend finite automata and transducers by allowing transitions to carry predicates and functions over rich alphabet theories, such as linear arithmetic. Therefore, these models extend their classic counterparts to operate over infinite alphabets, such as the set of rational numbers. Due to their expressiveness, symbolic automata and transducers have been used to verify functional programs operating over lists and trees, to prove the correctness of complex implementations of BASE64 and UTF encoders, and to expose data parallelism in computations that may otherwise seem inherently sequential. In this talk, I give an overview of what is currently known about symbolic automata and transducers as well as their variants. We discuss what makes these models different from their finite-alphabet counterparts, what kind of applications symbolic models can enable, and what challenges arise when reasoning about these formalisms. Finally, I present a list of open problems and research directions that relate to both the theory and practice of symbolic automata and transducers.

Automation of Correctness Checking in Education

Tatiana A. Andreyeva [0000-0002-1124-9499]

The A. P. Ershov Institute of Informatics Systems,
Siberian Branch of the Russian Academy of Sciences
6, Lavrentiev st., Novosibirsk, Russian Federation, 630090
ata@iis.nsk.su

Abstract. Having their origin in programming contests, now various systems for automated checking of solutions become useful not only in programming education. With their help, any teacher can organize a mini-contest or check own tests and students' works faster and easier. In order to attract a wider range of users, all processes of both checking and preparation should be automated.

The author extends the approach gained from own experience of using and creating the automated checking systems for the programming problems onto the automated checking of the non-programming problems.

The article discusses the structure of a problem, studies various approaches to automated checking, introduces problem complexes, suggests methods for creating accurate and consistent problem statements and check sets, and touches the automation of the preparation and checking processes.

Keywords: Automated Correctness Checking, Test Case Generation, Complex Information Systems.

Acknowledgements. The work is supported by the Russian Foundation for Basic Research grant RFBR № 18-07-01048

Introduction

Systems for the automated checking of solutions originate from programming contests. But today they become the means of automation of the teacher's work. They can be useful not only at programming classes. With their help, any teacher can organize a mini-contest or check own tests and students' works faster and easier.

But systems for automated checking also make additional demands to the contest problems. This requires a higher discipline from authors of all parts of a contest problem. To eliminate difficulties, the processes of preparation to the automated check-up should also be automated. The special system for the automated preparation of problem complexes can reduce the number of possible errors, ambiguities, and inconsistencies, especially if the problem's description, specifications, the check set, and the exemplar solution are created collegially.

To create full, explicit and consistent problems (not only for contests with automated checking of solutions but for all quizzes as well) it is necessary to realize that a

problem is a union of a statement, specifications and checking means. Thus we introduce the important notion of the *problem complex*.

To understand the structure of a proper system for the automated preparation of problem complexes, we thoroughly consider what a generalised problem, its solution and results are, how a solution method can be automatically checked, and how the number of check cases and the check approach depend on the amount and the type of the problem's variable and constant data, conditions, and restrictions.

Also, we touch some questions of the automated generation of check cases.

1 Problem, Solution and Result

The common-sense point of view that *to solve a problem* means *to find a correct answer* is obviously inacceptable since a wrong method of solution can entail a wrong answer. Moreover, the method can be the main point of the solution not only in programming but in other fields too. For example, in IQ tests [8], an often task is "find out the rule and then apply it".

Now, let a problem be represented by a triad $< \boldsymbol{D, C, R} >$, where $\boldsymbol{D}$ is known *data*, $\boldsymbol{C}$ is *conditions* necessary for creating a correct solution and obtaining a correct *result*, and $\boldsymbol{R}$ is unknown values to be found out during solving. Then the solving *method* $\boldsymbol{M}$ is a function, which creates the result $\boldsymbol{R}$ from the initial data $\boldsymbol{D}$ and conditions $\boldsymbol{C}$:

$$\boldsymbol{M: D \times C \rightarrow R}$$

Solution method is a sequence of inter-connected and inter-dependable components ($\boldsymbol{R_1}$, ... $\boldsymbol{R_N}$); each *stage* $\boldsymbol{R_i}$ is derived from stages $\boldsymbol{R_{|0...i-1|}}$ by a *step* of method $\boldsymbol{M_i}$. The whole method $\boldsymbol{M}$ is thus a sequence of solution steps ($\boldsymbol{M_1}$, ... $\boldsymbol{M_N}$).

The sequence ($\boldsymbol{R_1}$, ... $\boldsymbol{R_N}$) is not bound to be strictly linear, on the contrary, in most cases it only can be depicted by a graph structure where vertices are *stages* $\boldsymbol{R_i}$ ($i = 0 .. N$) and oriented edges are steps $\boldsymbol{M_i}$ ($i = 1 .. S$).

Each shift $\boldsymbol{M_i}$ from $\boldsymbol{R_i}$ to $\boldsymbol{R_{i+1}}$ can be converted into a subproblem and, thus, we can split the initial problem into several interconnected subproblems. The use of this point of view is studied in [4]. Here also lies the possibility of a partially parallel execution.

2 Correctness and Checking

Now that we agree that *to solve a problem* is *to find a correct solution method*, let us focus at the correctness of this method.

2.1 Equivalent Methods

A problem may have several correct solution methods. For example, in order to find the greatest common divisor (G. C. D.) of two natural numbers, one can either use the Euclidean algorithm or find all natural devisors for both numbers separately (by trying to divide them by each natural), compare these sets of divisors, and find the great-

est common one. Both of these methods correctly solve the initial problem but differ in the efficiency.[1]

Let us call two methods M' and M'' *equivalent* if being applied to the same input data D they produce the same result R.

2.2 What Is a Correct Method?

The most reliable way to check whether a method is correct is to prove that it solves the initial problem. Still, this way is too laborious to be exploited for mass checking. In education, we are in the situation of multiple methods to be checked in a short time and, thus, our aim is to reduce teacher's time and efforts. And the best way leading to automation of the checking process is to change the notion *correct*.

Suggesting students a problem in a test, teacher always has at least one "correct" solution method of this problem. This method we call *exemplar* and believe that it is actually correct. Here we should mention that proving the exemplar method's correctness might cause difficulties for teachers or contest organisers; and we have multiple examples when a wrong or incomplete exemplar solution was declared correct and the really correct ones were erroneously declined.

Nonetheless, henceforth we call method M *correct* if it is equivalent to the exemplar method M_{ex}. Thus we shift from checking of the correctness to checking of the equivalence.

To make the matter more intricate, there is a situation with multiple correct answers (refer to Section 3.1). In this case, the exemplar method should provide all possible correct results while the method under examination may produce only one of them. So, the equivalence should be not between two methods M and M_{ex} but between method M and only a sub-method of method M_{ex}.

A good way to eliminate teacher's errors can be proposed by a profoundly developed information system intended for automating the preparation of problem cases, which will be discussed later in Section 4.

2.3 Checking

Let us discuss the equivalence check-up suitable for the educational purposes.

Since equivalent methods must produce the same outcomes (see sections 2.1 and 2.2), it is sufficient to check whether the result produced by the method under examination coincides with the *exemplar outcome* which is the result produced by the exemplar method.

This approach ascends to the theory of so called *Black-box testing* introduced by Ashby in 1956 [6] and well developed for programming (see, for example, [6, 7 or

[1] If it is important that the solution method be a particular one, the author of the problem can shift the focus from the result to the method itself: not "Find the G.C.D. of two naturals" but "Describe the Euclidean algorithm of finding the G.C.D." In this new problem, what earlier was a method (one of several possible ones) became the only correct result.

13]); we shall try to adopt some of its methods for developing the theory of automated checking in education.

3 Practicable Input and Output

The exemplar outcome is the result of applying the exemplar method to some exemplar initial data. Now let us discuss the number of these.

3.1 Power of the Output

The number of all possible correct answers $|\boldsymbol{R}|$ is very important for our discussion of automation of checking. Here we only mention the possible variants. And the influence of the multiplicity of possible correct answers onto the checking process is discussed in Section 3.4.

$|\boldsymbol{R}| = \boldsymbol{0}$ means that the problem is stated erroneously. No correct answer is possible. Such problems must not appear in any test, quiz or contest.

$|\boldsymbol{R}| = \boldsymbol{1}$ means that there exists the only solution. In this case, checking is obvious and easy: it is sufficient to ascertain that the retrieved outcome coincides with the given exemplar one.

$|\boldsymbol{R}| > \boldsymbol{1}$ means that the problem has several correct answers. There can be three cases:

- $|\boldsymbol{R}|$ is finite. For example, "The anterior part of a shoulder is called a collar bone or a clavicle".
- $|\boldsymbol{R}|$ is infinite but denumerable. For example, "Any odd integer".
- $|\boldsymbol{R}|$ is infinite and non-denumerable. For example, "Any real value from the interval *[0 .. 1]*" or "Any point on the plane within the circle with the centre in *(0, 0)* and radius *1*".

Let us note that restrictions of the computer data representation obviously reduce the case of infinite (denumerable or non-denumerable) $|\boldsymbol{R}|$ to the case of finite $|\boldsymbol{R}| > \boldsymbol{1}$. The only difference between them is in approaches to automated checking.

3.2 Power and Dimension of the Input

Variables and Constants. Domain $\boldsymbol{D}$ of all known data can be divided into two parts: invariables $\boldsymbol{D_{inv}}$ and variables $\boldsymbol{D_{var}}$. *Invariable data* are stated in the problem's description and never change. On the contrary, *variable data* are provided during checking and can differ not only in value but in number too.

So, the power $|\boldsymbol{D_{var}}|$ can vary, and the power $|\boldsymbol{D_{inv}}|$ is constant. And the power of the whole domain $\boldsymbol{D}$ is their sum:

$$|\boldsymbol{D}| = |\boldsymbol{D_{var}}| + |\boldsymbol{D_{inv}}|.$$

No Variables. Obviously, there exist problems with no variable input. Almost all non-programming problems are like this. In this case, *power* $| \boldsymbol{D}_{var} | = 0$.

If such a problem is correctly stated, it must have the unique correct answer. As a rule, if there are several correct answers, it means that the problem description is not full or consistent.[2]

Variables as the Means to Split a Problem. In programming, any "good" algorithm has to be *mass-oriented*, i.e., it must be potent to solve not a single problem but a whole class of similar problems. Therefore, presence of a variable input is characteristic for programming problems.

Still, problems with variable input can be met not only in programming but in other fields too. For example, several variants of a quiz may include the same task with different numeric values.[3]

If there are several variables $V_1, \ldots V_N$, each of them having its own domain $\boldsymbol{D}_i$; then the whole variable domain $\boldsymbol{D}_{var}$ can be represented as a direct product of these domains:

$$\boldsymbol{D}_{var} = [\boldsymbol{D}_1 \times \ldots \times \boldsymbol{D}_N].$$

The dimension of the domain $\boldsymbol{D}_{var}$ is the sum of dimensions of $\boldsymbol{D}_1, \ldots \boldsymbol{D}_N$:

$$dim\ \boldsymbol{D}_{var} = dim\ \boldsymbol{D}_1 + \ldots + dim\ \boldsymbol{D}_N$$

We can make a *section* of the domain $\boldsymbol{D}_{var}$ by taking an actual value for each variable. With thus restricted input domain, the initial problem turns into a problem with no variables. Thus, we can reduce a problem with a variable input to a problem with the invariable input.

Constant and Known Number of Variables. Now let us consider one variable $\boldsymbol{V}_i$. The dimension of its domain can vary in a wide range from 1 to any value having the practical sense. Multidimensional input is common in programming, where the *dimension* denotes the number of variable's components.

Constant Bounds. As a rule, the lower and upper bounds for possible dimensions of a variable's domain are specified explicitly for all variables in the problem's description; they are known before the checking process has started. They can be considered as constants belonging to the invariable input $\boldsymbol{D}_{inv}$.

Variable Bounds. The bounds of a variable can also vary. Then they belong to $\boldsymbol{D}_{var}$ and must have bounds too. Let us call the changeable bound the *sub-bound*.

An example of such a situation is "N integers $A_1, \ldots, A_N$ are given ($1 < N < 100$)…" Here variable A consists of N components $A_1, \ldots A_N$ and, therefore, its dimension

[2] Various faults of problem statements are studied in [5].

[3] In such a case, each variant can be checked as an independent one and, therefore, can represent the single-answer case.

dim A = N. Variable *N* is the sub-bound of the current dimension, and *100* is the invariable upper bound common for all possible sub-bounds. Note that each A_i must have its own upper and lower limits too, but none are mentioned in this example.

So, the sub-bound is variable but becomes known when the check starts.

Indefinite Number of Variables. Now let us consider the case when the actual dimension of a variable's domain (the sub-bound) stays unknown until the end of checking.

Example is "No more than *100* integers are given…" Here we do not know how many components $A_1, \ldots A_N$ the actual input has. We can preliminary write all of them down and count them; then we will know the current sub-bound *N* (not given but calculated). Thus, we return to the case of a known sub-bound. On the other hand, we can process these components not using the value of *N* at all. The difference can be illustrated by cycles

for 1 to N do… and *do… until <the end is detected>*

Another example of the situation when the number of variables must be retrieved from the input is "A graph is specified by the list of its edges, which are pairs of vertex numbers".

Theoretically both the number of variants and their range can be infinite. Nonetheless, practically it is impossible due to restrictions of checking and computer representation.

3.3 Exemplar Input and Output

To check solutions automatically, we must have the exemplar input and the exemplar output.

Even in the case of finite domain ***D***, it is too generous to check all of its members (which are the problem's possible valid inputs). It is sufficient to apply the exemplar method only to some characteristic representatives.

Partial Cases. Domain ***D*** of all valid inputs can be split into equivalence classes.

Input data belong to the same equivalence class if being processed by the exemplar method they generate the same (or equivalent) outcomes. Each equivalence class is considered to be a *partial case*. Only one representative from each partial case is sufficient for the exemplar input.[4]

Unlike Beizer [7] or Myers [11, 12] who apply the equivalence partition to the domain of all possible inputs (and, thus, the affiliation with an equivalence class only shows whether a variable belongs to the valid range), we consider the domain of all valid inputs.

[4] For more details, see, for example, [7, 11 or 12].

Restriction of the Input. If domain ***D*** is infinite (refer to Section 3.2) then some (or even all) of the equivalence classes can be infinite too. If the number of classes is finite, getting one representative from each class forms a finite set of exemplar inputs.

Still, there can be infinite number of equivalence classes. To lessen this number, additional restrictions should be imposed on the domain of valid input data.

Automation. The partition of the domain ***D*** into equivalence classes can be done manually basing on the characteristics of the subject domain and the problem itself or automatically through the inner properties of the exemplar method. Still, the thorough discussion of this question is the topic of another article.

Check Cases. Having an exemplar method, one can trace all partial cases it processes. Since all inputs from an equivalence class are interchangeable, the representatives can be selected randomly.

Let a *check case* be a pair of some exemplar input and the corresponding exemplar output. We refer to a pack of check cases as a *check set*. Here we follow the analogy with *test cases* and *test sets* in programming.[5]

A check case is a set of points representing a section of domain ***D***. Therefore, a check case is a sub-problem of the initial problem where all variables in the ***D***, ***C*** and ***R*** have actual values.

3.4 Checking and Judging

The next step is to compare the acquired output with the corresponding exemplar output or the exemplar outcome.

|*R*| is Finite. If the problem admits only finite number of correct answers, the comparison can be easily performed by verifying the coincidence. In this case, the exemplar outcome should consist of one or several exemplar outputs. Let us also note that a poly-dimensional output brings almost no difference into the result-checking procedure.

|*R*| is Infinite. This case is more difficult. We cannot practically list all possible outputs; therefore, the "comparison" should mean performing a special checking formula, which depends on the type of the valid outputs. For example, we can ascertain that "a real Z belongs to the *[0..100]* interval" by checking that both $Z \geq 0$ and $Z \leq 100$ are true.

A poly-dimensional output can demand more complex formulas. For example, the result "a point on a plain with coordinates *(x, y)* belongs to the circumference with the center in *(0, 0)* and radius A" can be checked with the help of the pair of inequations

[5] Mostly, experts in programming (see, for example, [11, 12, 14, 15]) use the term *test suite* to name a pack of test cases. Still, *ISO/IEC/IEEE 24765:2010 International Standard – Systems and software engineering – Vocabulary* [10] does not mention this word at all. Instead, it uses the *test set* (3.3091). So we use *check set* as the synonym of *check suite* too.

$A^2 - e \le x^2 + y^2 \le A^2 + e$, where e is an admissible (and strictly specified in the problem's description) error.

If the author of the problem would rather avoid such difficulties, the problem's statement should be revised and the type of the output changed.

Judging. If the acquired output coincides with the example output (when $|\boldsymbol{R}|$ if finite) or meets the differently stated conditions (when $|\boldsymbol{R}|$ if infinite), the *check case result* is correct. Otherwise, it is incorrect.

After all check cases are processed, a judgment about the correctness of the whole method can be formed. And there are two ways for this.

The first way is the dichotomy "*all check case results are correct"* vs. *"at least one check case result is incorrect"*. The method is considered correct if and only if all its check case results are correct.

The second way is a gradation based upon the number of correct check case results. The metric for this gradation can be determined in various ways. For example, in an equipollent metric, each check case gives 1 point or *100/n* percents of the result. On the other hand, in a weighted metric, check cases make different contributions to the result. On this way, a method can be *more correct* or *less correct* than the other method, according to their metric values.

Judging about the correctness of a method under examination depends on the exemplar outcome and, therefore, on the accurateness of the exemplar method and of the coverage of all partial cases necessary for creating the exemplar input. For this reason, it is important to automate as many processes of preparation to automated checking as possible.

4 Automation

Let us look at a problem in the context of automated checking.

4.1 Problem Complexes

The *problem complex* should include (refer to [2] or [3]):

- ***Description*** of the problem
- ***Specifications*** of a valid input and output
- A ***check set***, which is a pack of exemplar *input-output* pairs
- An ***exemplar*** solution ***method***

The first and the second parts are "exterior". Contestants may see them. The third and the fourth parts, on the contrary, are for the inner use of checkers and judges only.

Problem's Description. The full description of a problem must contain, explicitly or implicitly, the following parts (for more details, refer to [5]):

- ***Introduction***
- ***Definitions***, agreements, terms, if necessary
- ***Statement***. A formalized presentation of the problem, its conditions and restrictions

- ***Task***. Requirements whose fulfilment means that the problem is solved
- ***Formats for the input and output data***
- ***Example*** of a correctly written down solution and result

Specifications. Input and output data specifications, restrictions and clauses are specified in the problem's description written in a natural language. A textual analysis can automatically extract the preliminary specification list, which should be revised manually [2, 3].

Check Set and *Exemplar method* were considered earlier in Sections 2.2 and 3.3.

4.2 Preparation of Problem Complexes

The process of preparing a problem complex is iterative: creating or changing each part (see Section 4.1) can impel changes in any other parts.

Stage 1. According to the original idea of the problem, the author of the problem's description

- defines restrictions R on all variables in use;
- makes a preliminary decomposition D^* of the valid input data domain $\boldsymbol{D}$ into equivalence classes showing all possible partial cases;[6]
- sets specifications S for the input and output data.

Stage 2. From specifications S and decomposition D^*, a check set CS is prepared. This can be done manually or automatically with the help of an automated test-preparing system (for more details, refer to [1]).

Stage 3. An exemplar solution ES is written (manually) and is debugged with the help of the check set CS.

To reduce the number of possible errors, it is recommended that problem complexes are created collegially. If two authors A_1 and A_2 write two different exemplar solutions ES_1 and ES_2 and use two check sets CS_1 and CS_2 for debugging, both of them fulfil stages 1 to 3, and then Stage 4 arises.

Stage 4. Two check sets are compared and combined. Both solutions ES_1 and ES_2 must be tested on the united check set $CS = CS_1 \cup CS_2$. If no cross-errors were detected, it is necessary to ascertain that this united check set agrees with the final decomposition D and meets the final specifications S. Most likely, the united check set CS will be superfluous; and, therefore, some surplus check cases should be excluded.

The 4th stage can also be useful for the individual preparation of a problem complex. The author's initial check set and the automatically generated check set can be treated as CS_1 and CS_2.

[6] On the very first stage, it is impossible to use the exemplar solution since it is not created yet.

4.3 Automated Systems for Preparation of Problem Complexes

Irrespectively to its actual correctness or erroneousness, the exemplar outcome is the base for judging about correctness of the method under examination (see Sections 2.2 and 3.4). Therefore, it is important to eliminate the possibility of errors in the exemplar method and the exemplar outcome. And here an *automated system for preparation of problem complexes* (ASPPC) can be of great use. Such systems are described in detail in [2, 3].

At Stage 1 (see Section 4.2), an ASPPC should

- Extract a preliminary set of specifications S_0 from the description of the problem by means of the textual analysis
- Check the consistency of specifications
- Extract possible information about bounds, exceptional points and so on from the problem's description and specifications S_0
- Construct a preliminary partition P of the valid data domain D into equivalence classes (basing on the specifications S_0 and additional information provided by the author(s) of the problem)
- Compare, join and intersect partitions P_1 and P_2

At Stage 2, an ASPPC should

- Create exemplar inputs basing on partition P
- Ascertain that the author's check set CS_0 covers partition P
- Verify that the check set CS_0 meets specifications S_0 and restrictions R

If the necessity to change the initial specification set S_0 is detected, the process of creating an exemplar check set should be started anew, now basing on the renewed specification set S_0'.

At Stage 3, with the exemplar outputs generated with the help of the exemplar solution (method), an ASSPPC should

- Check that the exemplar outputs meet specifications S (which is the final version of the specification set)

At Stage 4,

- Define the equivalent check cases
- Propose variants of reducing the joint check set

Conclusion

Our aim is to automate processes of preparing the problem complexes in any subject field, in order to make the automated checking easier and its use wider.

We have considered notions *checking* and *correctness* and have ascertained that not only programming problems but problems from other subject fields too can be checked automatically.

We have studied processes that constitute the preparation of a contest or a quiz and the check of their results and have shown which of these processes can be automated.

We have shown that automated systems make the preparation of problem complexes easier and more accurate, especially in case of co-working.

The future aims of our work are:

- To design means for the coverage analysis of the partitions created automatically from exemplar solutions,
- To develop the mathematical apparatus for operations with partitions of different types,
- To create means that can suggest additional partition variants basing on the analysis of the type, the power and the dimensions of the input data.

References

1. Andreyeva, T.: Automated generation of test sets. In: Science in the Modern Information Society IX: Proceedings of the conference. North Charleston, USA, 1-2.08.2016. pp. 110-112 (2016).
2. Andreyeva, T.: Automated preparation of problem complexes. In: Science today: Theoretical and practical aspects: Proceedings of the conference. Vologda, 27.12.2017. Part 1, pp. 25-26 (2018). Available at http:/volconf.ru/files/archive/01_27.12.2017.pdf
3. Andreyeva, T.: Automated preparation of problem complexes for programming contests. In: Science. Informatization. Technologies. Education XI: Proceedings of the international scientific-practical conference. Ekaterinburg, 26.02-2.03.2018. (in Russian)
   Available at the conference site http://nito.rsvpu.ru/files/nito2018/nito2018.pdf
4. Andreyeva, T.: Serial problems in programming. In: Perspectives of information systems V. Educational informatics section. Proceedings of the international conference. Novosibirsk, pp. 2-4 (2003) (in Russian)
5. Andreyeva, T.: Structure and classification of contest problems' texts. In: Computer instruments in education, 3-4, pp. 50-59 (2002). (in Russian).
   Available at http://ipo.spb.ru/journal/index.php?magazines/2002/34/e/
6. Ashby, W. R.: Introduction to cybernetics. Chapman & Hall (1956).
7. Beizer, B.: Black-box testing: Techniques for functional testing of software and systems. New York, NY, USA: John Wiley & Sons, Inc. (1995).
8. Eysenck, H. J.: Know your own I. Q. Penguin Books (1962).
9. Floyd, R. W.: Assigning meanings to programs. In: Mathematical Aspects of Computer Science. Proceedings of Symposium on Applied Mathematics. 19. American Mathematical Society. 19-32. (1967). ISBN 0821867288.
10. ISO/IEC/IEEE 24765:2010 International Standard. Systems and software engineering – Vocabulary. IEEE. DOI:10.1109/IEEESTD.2010.5733835
11. Myers, G. J.: The art of software testing. New York: John Wiley & Sons (1979).
12. Myers, G. J., Badgett, T., Sandler C.: The art of software testing. (3rd ed.) New York: John Wiley & Sons (2011).
13. Ponrod, C.: The study of black-box testing technique for collateral management system. Mahidol University Press (2014).
14. Singh, Y.: Software testing. Cambridge University Press. (2012). Chapter 1.3.4.
15. Spillner, A., Linz, T., Schaefer, H.: Software testing fundamentals: A study guide for the certified tester exam. (4th ed.) Rocky Nook Inc. (2014)

Two-Step Deductive Verification of Control Software Using Reflex*

Igor Anureev[1], Natalia Garanina[1,2], Tatiana Liakh[2,3], Andrei Rozov[2,3], Vladimir Zyubin[2,3], and Sergei Gorlatch[4]

[1] A. P. Ershov Institute of Informatics Systems, Acad. Lavrentieva prosp. 6, 630090 Novosibirsk, Russia
anureev@gmail.com, garanina@iis.nsk.su
[2] Novosibirsk State University, Pirogova str. 2, 630090 Novosibirsk, Russia
Institute of Automation and Electrometry, Acad. Koptyuga prosp. 1, 630090 Novosibirsk, Russia
{rozov,zyubin}@iae.nsk.su
[3] University of Muenster, Einsteinstr. 62, 48149 Münster, Germany
gorlatch@uni-muenster.de

Abstract. In this paper, we introduce a new verification method for control software. The novelty of the method consists in reducing the verification of temporal properties of a control software algorithm to the Hoare-like deductive verification of an imperative program that explicitly models time and the history of the execution of the algorithm. The method is applied to control software specified in Reflex — a domain-specific extension of the C language developed as an alternative to IEC 61131-3 languages. As a process-oriented language, Reflex enables control software description in terms of interacting processes, event-driven operations, and operations with discrete time intervals. The first step of our method rewrites an annotated Reflex program into an equivalent annotated C program. The second step is deductive verification of this C program. We illustrate our method with deductive verification of a Reflex program for a hand dryer device: we provide the source Reflex program, the set of requirements, the resulting annotated C program, the generated verification conditions, and the results of proving these conditions in Z3py – a Python-based front-end to the SMT solver Z3.

Keywords: control software · process-oriented languages · deductive verification · SMT solver · Reflex language · Z3.

1 Introduction

The increasing complexity of control systems used in our everyday life requires a reassessment of the design and development tools. Most challenging are safety-critical systems, where incorrect behavior and/or lack of robustness may lead

* This work has been supported by the Russian Ministry of Education and Science and the Russian Foundation for Basic Research (grant 17-07-01600).

to an unacceptable loss of funds or even human life. Such systems are widely spread in industry, especially, in chemical and metallurgical plants. Since behavior of control systems is specified in software, the study of control software is of great interest. Correct behavior under various environmental conditions must be ensured. In case of a hardware failure, e.g. plant damage or actuator fault, the control system must automatically react to prevent dangerous consequences. This is commonly referred to as fault tolerant behavior [1]. Because of the domain specificity, control systems are usually based on industrial controllers, also known as programmable logic controllers (PLCs), and specialized languages are used for designing control software.

PLCs are inherently open (i.e. communicate with external environment via sensors and actuators), reactive (have event-driven behaviour) and concurrent (have to process multiple asynchronous events). These features lead to special languages being used in the development of control software, e.g. the IEC 61131-3 languages [2] which are the most popular in the PLC domain. However, as the complexity of control software increases and quality is of higher priority, the 35 years old technology based on the IEC 61131-3 approach is not always able to address the present-day requirements [3]. This motivates enriching the IEC 61131-3 development model with object-oriented concepts [4], or developing alternative approaches, e.g. [5–8].

To address the restrictions and challenges in developing present-day complex control software, the concept of process-oriented programming (POP) was suggested in [9]. POP involves expressing control software as a set of interacting processes, where processes are finite state automata enhanced with inactive states and special operators that implement concurrent control flows and time-interval handling. Compared to well-known FSA modifications, e.g. Communicating Sequential Processes [10], Harel's Statecharts [11], Input/Output Automata [12], Esterel [13], Hybrid Automata [14], Calculus of Communicating Systems [15], and their timed extensions [16, 17], this technique both provides means to specify concurrency and preserves the linearity of the control flow at the process level. Therefore, it provides a conceptual framework for developing process-oriented languages suitable to design PLC software. The process-oriented approach was implemented in domain-specific programming languages, such as SPARM [18], Reflex [19], and IndustrialC [20]. These languages are C-like and, therefore, easy to learn. Translators of these languages produce C-code, which provides cross-platform portability. With their native support for state machines and floating point operations, these languages allow PLC software to be conveniently expressed.

The SPARM language is a predecessor of the Reflex language and is now out of use. IndustrialC targets strict utilization of microcontroller peripherals (registers, timers, PWM, etc.) and extends Reflex with means for handling interrupt. A Reflex program is specified as a set of communicating concurrent processes. Specialized constructs are introduced for controlling processes and handling time intervals. Reflex also provides constructs for linking its variables to physical I/O

signals. Procedures for reading/writing data through registers and their mapping to the variables are generated automatically by the translator.

Reflex assumes scan-based execution, i.e. a time-triggered control loop, and strict encapsulation of platform-dependent I/O subroutines into a library, which is a widely applied technique in IEC 6113-3 based systems. To provide both ease of support and cross-platform portability, the generation of executable code is implemented in two stages: the Reflex translator generates C-code and then a C-compiler produces executable code for the target platform. Reflex has no pointers, arrays or loops. Despite its very simple syntax, the language has been successfully used for several safety-critical control systems, e.g., control software for a silicon single-crystal growth furnace [21]. Semantic simplicity of the language together with the continuing practical applicability makes Reflex attractive for theoretical studies.

Currently, the Reflex project is focused on design and development tools for safety-critical systems. Because of its system independence Reflex easily integrates with LabVIEW [22]. This allows to develop software combining event-driven behavior with advanced graphic user interface, remote sensors and actuators, LabVIEW-supported devices, etc. Using the flexibility of LabVIEW, a set of plant simulators was designed for learning purposes [23]. The LabVIEW-based simulators include 2D animation, tools for debugging, and language support for learning of control software design. One of the results obtained in this direction is a LabVIEW-based dynamic verification toolset for Reflex programs. Dynamic verification treats the software as a black-box, and checks its compliance with the requirements by observing run-time behavior of the software on a set of test-cases. While such a procedure can help detect the presence of bugs in the software, it cannot guarantee their absence [24].

Unlike dynamic verification, static methods are based on source code analysis and are commonly recognized as the only way to ensure required properties of the software. It is therefore very important to adopt static verification methods for Reflex programs.

In this paper, we propose a method of deductive verification of Reflex programs. The original two-step scheme of the method allows us to reduce the verification of temporal properties of a control algorithm written in Reflex to the Hoare-like deductive verification of a C program that explicitly models time and the history of the execution of the algorithm.

The paper has the following structure. In Section 2, we describe the language for specifying of temporal properties of Reflex programs and an example of a Reflex program controlling a hand dryer with its properties. Section 3 presents the algorithm of transforming an annotated Reflex programs into a very restricted subset of annotated C programs called C-projections of Reflex programs. We illustrate this algorithm by the example of the C-projection of the dryer-controlling program. Rewriting an annotated Reflex program into its C-projection is the first step of our deductive verification method. The second step — generation of verification conditions for C-projection programs of this subset — is defined in Section 4. Examples of verification conditions for the C-

projection of the dryer-controlling program illustrate the rules of this generation. In the concluding Section 5, we discuss the features of our method and future work.

2 Specification of properties of Reflex programs

Our verification method reduces the verification of Reflex programs to the verification of C-projection programs.A Reflex program, together with its requirements for verification, is translated into an equivalent C-projection program and a corresponding set of properties. In this section, we define the specification method for the properties of Reflex programs. This method is illustrated with an example Reflex program for a hand dryer controller.

We specify properties of Reflex programs using two kind of languages: an annotation language and an annotating language. The *annotation language* is a language of logic formulas that describe program properties. These formulas are called *annotations*. The *annotating language* is a markup language for attributing annotations to a program. Constructs of this language are called *annotators*. A program extended with annotators is an *annotated program*.

Annotations of Reflex programs are formulas of a many-sorted first-order logic. The specific formula syntax in the example uses the language of the python-based front-end Z3py [25] to the SMT solver Z3 [26] used in deductive verification of the resulting C-projection programs.

Temporal properties of Reflex programs can be expressed in the annotations. The discrete-time model used in the annotations is based on the periodicity of interaction between a Reflex program and its object under control. A Reflex program and its controlled object interact via input and output ports associated with the program variables. Every time-triggered control loop the program reads input ports and then writes the values to the corresponding variables. Changing a variable value as a result of writing to an input port is called its *external update*. At the end of control loop, the program writes new values to output ports. Writing values from input ports to variables and reading values from variables to output ports occur periodically with a fixed period (program cycle) specified in milliseconds. Time in the annotations is modeled by the implicit variable *tick* (which is not used in Reflex programs explicitly) specifying the number of program cycles. Thus, *tick* is an analogue of the global clock, counting the number of interactions of the Reflex program with its controlled object. One tick of the clock corresponds to one program cycle.

Each program variable x is interpreted in the annotations as an array in which indexes are values of *tick*, and elements are values of x associated with *tick*. Thus, in the annotation context, x stores a history of its changes. We denote a set of annotations by F, such that $f \in F$ is an annotation specifying some Reflex program property.

The annotating language for Reflex programs includes three kinds of annotators. *The invariant annotator* `INV` f`;` specifies that the property f must be true before each program cycle. *The initial condition annotator* `ICON` f`;` specifies that

the property f must be true before the first program cycle. *The external condition annotator* `ECON` f`;` constrains external updates: the property f must be true after each external update.

Let us illustrate our approach by using a simple example of a program controlling a hand dryer like those often found in public restrooms (Fig. 1, Listing 1).

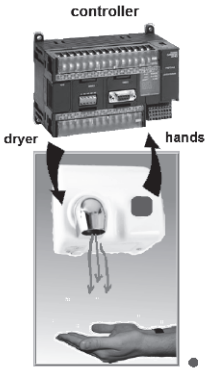


Fig. 1. Hand Dryer

Here, the program uses the input from an infrared sensor, indicating presence of hands under the dryer and it controls the fan and heater with a joint output signal. The first basic requirement is that the dryer is on while hands are present and it turns off automatically otherwise. Trivial at first sight, the task becomes complicated because of discontinuity of the input signal caused by the users rubbing and turning their hands under the dryer. To avoid erratic toggling of the dryer heater and fan, the program should not react to brief interruptions in the signal, and the actuators should only be turned off once the sensor reading is a steady "off". The control algorithm can only meet this requirement by measuring the duration of the off state of the sensor. In this case, a continuous "off" signal longer than a certain given time (for example, 1s) would be regarded as a "hands removed" event. The second requirement is more simple and formulated as 'dryer never turns on spontaneously'. These two requirements (specified by the formulas p_1 and p_2 below) we will verify to demonstrate the proposed approach.

```
    PROGR HandDryerController {
/* ========================== */
/* == ANNOTATIONS:           */
/* INV inv;                  */
/* ICON icon;                */
/* ECON econ;                */
/* == END OF ANNOTATIONS     */
      TACT 100;
      CONST ON 1;
      CONST OFF 0;
    /*============================*/
    /* I/O ports specification    */
    /* direction, name, address,  */
    /* offset, size of the port   */
    /*============================*/
      INPUT  SENSOR_PORT  0 0 8;
      OUTPUT ACTUATOR_PORT 1 0 8;

    /*============================*/
    /* processes definition       */
    /*============================*/
      PROC Ctrl {
    /*===== VARIABLES ============*/
        BOOL hands = {SENSOR_PORT[1]} FOR ALL;
        BOOL dryer = {ACTUATOR_PORT[1]} FOR ALL;

    /*===== STATES ===============*/
        STATE Waiting {
          IF (hands == ON) {
```

```
          dryer = ON;
          SET NEXT;
        } ELSE dryer = OFF;
      }
      STATE Drying {
        IF (hands == ON)
          RESET TIMEOUT;
          TIMEOUT 10
            SET STATE Waiting;
      }
    } /* \PROC */
}  /* \PROGRAM */
```

Listing 1. Hand dryer example in Reflex

In Reflex programs, the `PROGR` construct specifies the name and body of the program. The annotators are added at the beginning of the program body as the special kind of comments. In our case the annotators are `INV inv;`, `ICON icon;`, and `ECON econ;`, where `inv`, `icon`, and `econ` are annotations defined below. The `TACT` construct specifies the number of milliseconds corresponding to one program cycle. The `CONST` construct is used to specify program constants. Constructs `INPUT` and `OUTPUT` describe the input and output ports, respectively. Program variables are specified by variable declarations. For example, the variable declaration `BOOL hands = SENSOR_PORT[1] FOR ALL;` associates the boolean variable `hands` with the first bit of the port `SENSOR_PORT` and specifies that all processes can use this variable. The `PROC` construct is used to describe processes of the program. Our example program has one process `Ctrl` (controller) that controls a hand dryer, i.e. its fun and heater. The `STATE` construct specifies process states. Process `Ctrl` can be in two states `WAITING` and `DRYING`. Actions executed by the process in a state are described in the body of that state by statements and operators. In addition to C-statements and operators, there are Reflex-specific ones. Each process has its own time counter (local clock), which is also counted in ticks (the number of program cycles). Statement `RESET TIMEOUT;` resets the local clock of the process. Statement `TIMEOUT` x *stm*`;` launches the execution of statement *stm* when the local clock is equal to x. Statement `SET NEXT;` moves the process to the next state in the text of the program, and statement `SET STATE s;` sets the process to the state s. These two statements also reset the local clock of the process.

The initial condition `icon` of the form (in the format of formulas in Z3py [25])

$$And(Or(dryer[0] == 0, dryer[0] == 1), Or(hands[0] == 0, hands[0] == 1))$$

specifies that variables *dryer* and *hands* can only have values 0 or 1. The external condition `econ` of the form

$$Or(hands[tick] == 0, hands[tick] == 1)$$

expresses the fact that external updates of *hands* return 0 or 1.

Invariant `inv` of the form $And(p_1, p_2, ap)$ includes properties p_1 and p_2 which specify the desirable behaviour of the program and the conjunction ap of auxiliary properties necessary to verify them. These auxiliary properties are as follows: 1) the values of the program constants are equal to their predefined values,

2) counter *tick* is non-negative, 3) all previous and current values of variables *hands* and *dryer* are 0 or 1, 4) the current values of the latter variables are the same as their previous values (since they have not yet been modified by external updates), 5) the dryer can only be in two states `WAITING` and `DRYING`, and 6) the dryer in state `DRYING` is always on. We omit the notation for *ap* because it is rather cumbersome.

Property p_1 of the form

$$ForAll(i, Implies(And(0 <= i, i < tick),\\ Implies(And(Implies(i > 0, hands[i-1] == 0), hands[i] == 1),\\ dryer[i] == 1)))$$

refines the first hand-dryer requirement that the dryer is turned on ($dryer[i] = 1$) no later than 100 millisecond (1 tick) after the appearance of hands.

Property p_2 of the form

$$ForAll(i, Implies(And(0 <= i, i < tick - 1),\\ Implies(And(dryer[i] == 0, hands[i+1] == 0), dryer[i+1] == 0)))$$

corresponds to the second requirement that the dryer never turns on spontaneously.

In the next section, we present the method of rewriting an annotated Reflex program to the annotated C-projection to generate the verification conditions and subsequently check them with a theorem proving tool which can handle the many-sorted first-order logic. We apply this method to the Reflex program describing the hand dryer controller.

3 Rewriting annotated Reflex programs into C-projections

Reflex programs and their C-like projections share the same annotation language. The annotating language for C-projections of Reflex programs includes four annotators. *The assume annotator* `ASSUME f;` specifies that f is supposed to be true at the location of this annotator in the program. *The assert annotator* `ASSERT f;` states that f must be true at the location of this annotator in the program. *The invariant annotator* `INV l f;` is a special variant of the named assert annotator with the name l which is processed by our verification condition generator in a special way. *The function annotator* `REQUIRES` P_f`;` `ENSURES` Q_f`;` must be placed directly after the function prototype $t\ f\ (t_1\ x_1,\ \ldots,\ t_n x_n)$. The function prototypes are used to call functions written in other programming languages in Reflex programs. This annotator specifies the precondition P_f and postcondition Q_f of the function f. Formulae P_f and Q_f depend on $x_1, \ldots, x_n$. Postcondition Q_f also depends on the special variable ret_f which stores the value returned by f. The variables $x_1, \ldots, x_n$ and ret_f are considered to be global variables of the C-projection program.

The C-projection of the Reflex program for a hand dryer controller reads as follows:

```
#define TACT 100
#define ON 1
#define OFF 0
#define STOP_STATE 0
#define ERROR_STATE 1
#define Ctrl_Waiting 2
#define Ctrl_Drying 3

int Ctrl_state;
int Ctrl_clock;
int tick;
int hands[];
int dryer[];

inline void init() {
  tick = 0;
  Ctrl_state = Ctrl_Waiting;
  Ctrl_clock = 0;
  ASSUME icon;
}

inline void Ctrl_exec() {
  switch (Ctrl_state) {
    case Ctrl_Waiting:
      if (hands[tick] == ON) {
        dryer[tick] = ON;
        Ctrl_clock = 0;
        Ctrl_state = Ctrl_Drying;
      }
      else
        dryer[tick] = OFF;
      break;
    case Ctrl_Drying:
      if (hands[tick] == ON) {
        Ctrl_clock = 0;
        Ctrl_state = Ctrl_Drying;
      }
      if (Ctrl_clock >= 10) {
        Ctrl_clock = 0;
        Ctrl_state = Ctrl_Waiting;
      }
      break;
  }
}

void main() {
  init();
  for (;;) {
    INV lab inv;
    havoc hands[tick];
    ASSUME econ;
    Ctrl_exec();
    Ctrl_clock = Ctrl_clock + 1;
    tick = tick + 1;
    hands[tick] = hands[tick-1];
    dryer[tick] = dryer[tick-1];
  }
}
```

Listing 2. Hand dryer example in C-projection

This program is the result of applying program transformation rules that are used for generating an equivalent program that must include the following constructs which replace the source Reflex constructs.

The macro constant `TACT` specifying the time of the program cycle replaces the `TACT` construct. Reflex constants (for example, `ON` and `OFF`) are replaced by macro constants as well. The macro constants `STOP_STATE` and `ERROR_STATE` encode the stop state (specifying that the program terminates normally) and the error state (specifying that the program terminates with an error). For each program process `p` and for each state `s` of this process, the macro constant `s_p` encodes this state. The variable `tick` specifies the global clock. For each program process `p`, the variables `p_state` and `p_clock` specify the current state and the current value of the local clock of the process `p`. Like `tick`, these variables are also implicit variables of the Reflex program, and so they can be found in its annotations. The type `t` of each Reflex variable `x` is replaced by the dynamic array type `t[]`.

Function `init()` initializes the program processes. It sets the global clock and all local clocks to 0, sets all processes to their initial states and imposes restrictions on the initial values of Reflex variables, using the assume annotator `ASSUME f` (for the hand-dryer program `ASSUME icon`).

For each program process `p`, function `p_exec` specifies the actions of the process `p` during the program cycle. The body of function `p_exec` represents the switch statement where labels are macro constants coding states of the process `p_exec`. All Reflex-specific statements and operators in bodies of process states are replaced by C constructs in accordance with their semantics.

The infinite loop `for(;;)` specifying the actions of all processes during the program cycle is the last statement of the resulting program. Its body starts with the invariant annotator `INV lab inv;` specifying the invariant `inv` of the Reflex program. The next fragment `havoc hands[tick]; ASSUME econ;` specifies external updates of Reflex variables (in our case, `hands`) and the constraint `econ` for them. We add the special statement `havoc x;` [27] to the standard C language in order to model assigning an arbitrary value to the variable x. The third fragment is a sequence of calls of the functions `p_exec()` for each program process `p`. The next fragment increments the values of global clock and all local clocks. The last fragment specifies that values of Reflex variables are preserved after incrementing the global time and before executing external updates. For the hand-dryer program, this fragment is `hands[tick] = hands[tick-1]; dryer[tick ] = dryer[tick-1];`.

The definition of the transformational semantics of a Reflex program (the rules for its transformation into C projection) and proving transformation correctness (equivalence of the Reflex program and its C projection) are beyond the scope of this paper. The equivalence means functional equivalence of the Reflex program and its C projection, where the inputs of both programs are the external updates vector for each Reflex variable, and the outputs are the vector of values for each Reflex variable, as well as the current process states and the values of global clock and local clocks. It is based on the operational semantics of Reflex programs, their C projections, and annotators of both annotation languages.

Thus, we reduce the verification of Reflex programs to the verification of programs of a very restricted subset of C extended by the havoc statement. Next we describe the rules of generating the verification conditions for programs of this subset. These verification conditions can further be checked by some theorem proving tool that can handle many-sorted first-order logic.

4 Generating Verification Conditions for C-projections of Reflex programs

Like many other deductive verification engines, such as FramaC [28], Spark [29], KeY [30], Dafny [31], etc., our algorithm for generating verification conditions implements a predicate transformer. We use Z3 to prove such verification conditions. Let us consider the features of its implementation especially taking into account the fact that it is applied to a program which is an infinite loop and some variables of this program are externally changed at each iteration of the loop. The algorithm $sp(A, P)$ recursively calculates the strongest postcondition [32] expressed in the many-sorted first-order logic for program fragment A and precondition P. It starts with the entire program and the precondition $True$. Its output is the set of verification conditions saved in the variable vcs. The algorithm uses service variables $vars$ and $reached$. Variable $vars$ stores information about variables and their types as a set of pairs of the form $x : t$, where x is a variable, and t is its type. Variable $reached$ stores the set of names of invariant annotators that have been reached by the algorithm. It is used to ensure termination of the algorithm. The initial values of these variables are empty sets.

We define the generation algorithm sp as the ordered set of equalities of the form $sp(A, P) = [a_1;\ \ldots;\ a_n;\ e]$. This notation means that the actions a_1, …, a_n are sequentially executed before the expression e is computed. Every action a_i of the form $v\ +=\ S$ adds the elements of the set S to the set v. The equality $sp(A, P) = e$ is an abridgement for $sp(A, P) = [e]$.

We use the following notation in the algorithm definition. Let $array(t)$ denote the array type with the elements of type t. Let expression e have type t, $\{x : t, y : array(t)\} \subseteq vars$, $\{z : t, v : t\} \cap vars = \emptyset$ for each t, and e' be the result of conversion of C expression e to a Z3py expression. Function $Store(a, i, v)$ is the array update function from Z3 language.

Since the syntax of C-projections of annotated Reflex programs is very restricted, algorithm sp has the following compact form:

1. $sp(t\ f(t_1\ x_1,\ \ldots,\ t_n\ x_n);,\ P) =$
   $[vars\ +=\ \{x_1 : t_1, \ldots, x_n : t_n, ret\_f : t\};\ P]$;
2. $sp(t\ x;,\ P) = [vars\ +=\ \{x : t\};\ P]$;
3. $sp(\#define\ c\ e;,\ P) = [vars\ +=\ \{c : t\};\ And(P, c == e')]$;
4. $sp(havoc\ y[i];,\ P) =$
   $[vars\ +=\ \{z : t, v : t\};\ And(P(y \leftarrow z), y == Store(z, i, v))]$;
5. $sp(havoc\ x;,\ P) = [vars\ +=\ \{z : t\};\ And(P(x \leftarrow z), x == z)]$;
6. $sp(x[i]\ =\ e;,\ P) =$
   $[vars\ +=\ \{z : array(t)\};\ And(P(y \leftarrow z), y == Store(z, i, e'(y \leftarrow z)))]$;

7. $sp(x\ =\ e;,\ P) = [vars\ +=\ \{z : t\};\ And(P(x \leftarrow z), x == e'(x \leftarrow z))]$;
8. $sp(\{B\},\ P) = sp(B,\ P)$;
9. $sp(if\ (e)\ B\ else\ C,\ P) =$
$Or(sp(B, And(P, econv(e))), sp(C, And(P, Not(econv(e)))))$;
10. $sp(switch\ (e)\ l_1 :\ B_1\ break;\ \ldots\ l_n :\ B_n\ break;,\ P) =$
$Or(sp(B_1, And(P, e' == l_1)), \ldots, sp(B_n, And(P, e' == l_n)),$
$And(P, e'! = l_1, \ldots, e'! = l_n))$;
11. $sp(for(;;)\ B,\ P) = sp(B\ for(;;)\ B,\ P)$;
12. $sp(x\ =\ f(e_1,\ \ldots,\ e_n);,\ P) =$
$sp(x_1\ =\ e_1;\ \ldots\ x_n\ =\ e_n;\ ASSERT\ P_f;\ havoc\ ret\_f;$
$ASSUME\ Q_f;\ x\ =\ ret\_f;,\ P)$;
13. $sp(ASSUME\ e;,\ P) = And(P,\ e)$;
14. $sp(ASSERT\ e;,\ P) = [vcs+ = \{Implies(P, e)\};\ And(P, e)]$;
15. if $l \notin reached$, $sp(INV\ l\ e;\ A,\ P) =$
$[reached\ +=\ \{l\};\ vcs\ +=\ Implies(P, e);\ sp(A, e)]$;
16. if $l \in reached$, $sp(INV\ l\ e;\ A,\ P) = [vcs\ +=\ Implies(P, e);\ e]$;
17. $sp(s\ A,\ P) = sp(A,\ sp(s,\ P))$.

This algorithm terminates because *sp* recursively reduces the input program in all cases except $for(;;)$ (case 11), and due to case 16 the algorithm can pass the invariant annotator at the begin of body of $for(;;)$ only once.

The computation of verification conditions for the trace of the annotated hand dryer program (Listing 2) starting at the point `#define TACT 100` and ending at the point `INV lab inv;` results in

- $vcs = \{Implies(And(true, TACT == 100, ON == 1, OFF == 0,$
$STOP\_STATE == 0, ERROR\_STATE == 1, Ctrl\_WAITING == 2,$
$Ctrl\_DRYING == 3, tick == 0, init\_state == INIT\_WAITING,$
$init\_clock == 0, dryer[0] == 0, icon), inv)\}$;
- $vars = \{TACT : int, ON : int, OFF : int, STOP\_STATE : int,$
$ERROR\_STATE : int,\ Ctrl\_WAITING : int,\ Ctrl\_DRYING : int,$
$Ctrl\_state : int, Ctrl\_clock : int, dryer : array(int), hands : array(int),$
$tick : int, tick\_1 : int, Ctrl\_state\_1 : int, Ctrl\_clock\_1 : int\}$;
- $reached = \{lab\}$.

Here x_i, where i is a natural number, is a fresh variable generated by algorithm *sp* in the case of the assignment of the form $x = \ldots$ or $x[\ldots] = \ldots$.

Other seven verification conditions starting at the point `INV lab inv;` and ending at the same point and corresponding to different branches of the switch statement and if statements are generated likewise. All generated verification conditions are successfully proved in Z3py.

The generation of verification conditions for C-projections of annotated Reflex programs and proving them complete the description of our two-step method of deductive verification for Reflex programs.

Currently, we prepare for publication a description of the transformational semantics of Reflex programs (including a formal proof of its correctness) and a formal proof of the soundness of the axiomatic semantics of C projections w.r.t.

their operational semantics. Software tools automating the steps of the method are being developed on their basis.

5 Discussion and Conclusion

In this paper we propose a two-step method of deductive verification of Reflex programs. This method includes the annotation and annotating languages for Reflex programs, the algorithm for transforming an annotated Reflex program into the annotated program written in restricted C (the C-projection of the Reflex program), the annotating language, and the algorithm of generating verification conditions for C-projections of Reflex programs. Our method can be applied to the so-called pure Reflex programs that do not contain definitions of functions written in other languages.

In practice, Reflex programs often include definitions and calls of C functions. We can extend our method to this more general case of Reflex, because such programs must also include a prototype for each C function. Verification of such Reflex programs is reduced to separate verification of definitions of C functions and a pure Reflex program extended by calls of functions. These functions are considered black boxes and their prototypes are annotated with preconditions and postconditions treated as specifications of these black boxes. The definitions of C functions can be verified by any C program verification method or tool. For verification of Reflex programs with function calls, we use our two-step method.

Our verification method has several remarkable properties. Firstly, it models the interaction between a Reflex program and its controlled object through the input and output ports associated with the program variables. The havoc statement in the C-projection of the Reflex program allows to represent writing values from input ports to variables. These external variable updates are constrained by the assume annotator. Checking values read from variables to output ports is specified by the assert and invariant annotators. Secondly, this method reduces the verification of some time properties of Reflex programs to Hoare-like deductive verification by explicitly modeling time in C-projections of Reflex programs with variables which specify the global clock, local clocks of program processes and history of values of Reflex variables. Thirdly, our verification conditions generation algorithm can handle infinite loops intrinsic to control systems.

There are several directions for further development of the method. We plan to extend it to the textual languages of the IEC 61131-3 family. Like Reflex, these languages are used for programs that interact with the controlled object only between program cycles. The other research direction is to investigate new temporal properties for which verification can also be reduced to Hoare-like deductive verification. Especially, we are interested in temporal aspects associated with the histories of values of process states and process clocks, which would allow to evaluate the performance of control sofware algorithms. Explicit time modeling in Reflex annotations is not a very natural way to represent the time properties of Reflex programs. We plan to use temporal logics (LTL, CTL and

MTL) and their extensions to describe these properties and develop an algorithm for translating such descriptions into formulas with explicit time modeling. To make this task feasible, we plan to use specialized ontological patterns [33] instead of arbitrary formulas of these logics. In addition to Z3 solver, we intend to use in our method other provers and solvers in order to extend the class of verifiable properties. In particular, Z3 solver cannot prove that dryer will work for at least 10 seconds after hands have been removed because this property requires advanced induction. The interactive theorem prover ACL2 [34] with advanced induction schemes is a good candidate to solve this problem. Finally, we plan to consider new case studies on control software algorithms.

References

1. Blanke, M., Kinnaert, M., Lunze, J., Staroswiecki, M.: Diagnosis and Fault-Tolerant Control. 2nd edn. Springer-Verlag Berlin, Heidelberg (2006)
2. IEC 61131-3: Programmable controllers Part 3: Programming languages. Rev. 2.0. Intern. Electrotechnical Commission Std. (2003)
3. Basile, F., Chiacchio, P., Gerbasio, D.: On the Implementation of Industrial Automation Systems Based on PLC. IEEE Transactions on Automation Science and Engineering **4**(10), 990–1003 (2013)
4. Thramboulidis, K., Frey, G.: An MDD Process for IEC 61131-based Industrial Automation Systems. In: 16th IEEE Intern. Conf. on Emerging Technologies and Factory Automation (ETFA11), pp. 1–8. Toulouse, France (2011)
5. IEC 61499: Function Blocks for Industrial Process Measurement andControl Systems. Parts 1 – 4. Rev. 1.0. Intern. Electrotechnical Commission Std (2004/2005)
6. Wagner, F., Schmuki, R., Wagner, T., Wolstenholme, P.: Modeling Software with Finite State Machines. Auerbach Publications, USA, Boston, MA (2006)
7. Samek, M.: Practical UML statecharts in C/C++: event-driven programming for embedded systems. 2nd edition. Newnes, Oxford (2009)
8. Control Technology Corporation. QuickBuilder^{TM}Reference Guide. 2018, https://controltechnologycorp.com/docs/QuickBuilder_Ref.pdf. Last accessed 20 Jan 2019
9. Zyubin, V. E.: Hyper-automaton: A Model of Control Algorithms. In : Proceedings of the IEEE Intern. Siberian Conf. on Control and Communications (SIBCON-2007), pp. 51–57. The Tomsk IEEE Chapter & Student Branch, Tomsk, Russia (2007)
10. Hoare, C. A. R.: Communicating Sequential Processes. Prentice-Hall Int. (1985).
11. Harel, D.: Statecharts: a Visual Formalism for Complex Systems. Science of Computer Programming **8**(3), 231–274 (1987)
12. Lynch, N., Tuttle, M.: An Introduction to Input/Output Automata. CWI Quarterly **2**(3), 219–246 (1989)
13. Berry, G.: The Foundations of Esterel,.In: Proof, Language and Interaction: Essays in Honour of Robin Milner, pp. 425–454. MIT Press, Foundations of Computing Series (2000)
14. Henzinger, T.A.: The Theory of Hybrid Automata. In: Inan M.K., Kurshan R.P. (eds) Verification of Digital and Hybrid Systems, NATO ASI Series (Series F: Computer and Systems Sciences), vol. 170. pp. 265–292. Springer, Berlin, Heidelberg (2000)

15. Milner, R.: Communication and Concurrency. Series in Computer Science. Prentice Hall, New Jersey (1989).
16. Kaynar, D. K., Lynch, N., Segala, R., Vaandrager, F.: Timed I/O Automata: A Mathematical Framework for Modeling and Analyzing Real-Time Systems. In: 24th IEEE Intern. Real-Time Systems Symposium (RTSS'03), pp. 166–177. IEEE Computer Society Cancun, Mexico (2003)
17. Kof, L., Schätz, B.: Combining Aspects of Reactive Systems. In: Proc. of Andrei Ershov Fifth Int. Conf. Perspectives of System Informatics, pp. 239–243. Novosibirsk (2003)
18. Zyubin, V.: SPARM Language as a Means for Programming Microcontrollers. Optoelectronics, Instrumentation, and Data Processing, **2**(7), 36–44, (1996)
19. Liakh, T. V., Rozov, A. S., Zyubin, V. E.: Reflex Language: a Practical Notation for Cyber-Physical Systems. System Informatics. **12**(6), 85–104 (2018)
20. Rozov A.S., Zyubin V.E.: Process-oriented programming language for MCU-based automation. In: Proc. of the IEEE Intern. Siberian Conf. on Control and Communications, pp. 1–4. . The Tomsk IEEE Chapter Student Branch, Tomsk, Russia (2013)
21. Bulavskij, D., Zyubin, V., Karlson, N., Krivoruchko, V., Mironov, V.: An Automated Control System for a Silicon Single-Crystal Growth Furnace. Optoelectronics, instrumentation, and data processing **2**(5), 25–30 (1996)
22. Travis, J., Kring, J.: LabVIEW for Everyone: Graphical Programming Made Easy and Fun. 3rd Edition. Prentice Hall PTR, Upper Saddle River, NJ, USA (2006)
23. Zyubin, V.: Using Process-Oriented Programming in LabVIEW. In: Proc. of the Second IASTED Intern. Multi-Conference on "Automation, control, and information technology": Control, Diagnostics, and Automation, pp. 35–41. Novosibirsk (2010)
24. Randell, B.: Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee, p. 16. Brussels, Scientific Affairs Division, NATO, Rome, Italy (1970)
25. Z3 API in Python, https://ericpony.github.io/z3py-tutorial/guide-examples.htm Last accessed 20 Jan 2019
26. Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: TACAS 2008: Tools and Algorithms for the Construction and Analysis of Systems, LNCS, vol. 4963, pp. 337–340 (2008)
27. Barnett, M., Chang, B.-Y.E., DeLine, R., Jacobs, B., Leino, K.R.M.: Boogie: A Modular Reusable Verifier for Object-Oriented Programs. In: Proc. of the 4th Intern. Conf. on Formal Methods for Components and Objects, LNCS, vol. 4111, pp. 364–387 (2005)
28. FramaC Homepage, https://frama-c.com/
29. Spark Pro Homepage, https://www.adacore.com/sparkpro
30. The KeY project Homepage, https://www.key-project.org/
31. Dafny Homepage, https://www.microsoft.com/en-us/research/project/dafny-a-language-and-program-verifier-for-functional-correctness/
32. Dijkstra, E.W., Scholten, C.S.: Predicate Calculus and Program Semantics. Springer-Verlag (1990)
33. Garanina, N., Zyubin, V., Lyakh, V., Gorlatch, S.: An Ontology of Specification Patterns for Verification of Concurrent Systems. In: New Trends in Intelligent Software Methodologies, Tools and Techniques. Proc. of the 17th Intern. Conf. SoMeT-18. Series: Frontiers in Artificial Intelligence and Applications, Amsterdam: IOS Press, pp. 515-–528. (2018)
34. ACL2 Homepage, http://www.cs.utexas.edu/users/moore/acl2/

A Metamodel-based Approach for Adding Modularization to KeYmaera's Input Syntax*

Thomas Baar[1] [0000−0002−8443−1558]

Hochschule für Technik und Wirtschaft (HTW) Berlin, Department of Engineering I, Wilhelminenhofstraße 75A, 12459 Berlin, Germany thomas.baar@htw-berlin.de

Abstract. The theorem prover KeYmaera allows (1) to describe Cyber-Physical Systems (CPSs) in terms of a Hybrid Program (HP), (2) to specify properties for the defined system, and (3) to formally verify these properties in a tailored logic called Differential Dynamic Logic (DDL).
The syntax of Hybrid Programs is rather poor and covers only the most basic program statements, such as assignment, test, sequential execution, and iteration. The decision to keep the syntax of HPs very simple has different consequences: An advantage is that also the verification calculus can be kept relatively simple. On the downside we have that even small programs are hard to understand and that the programmer is forced to program using a copy-and-paste style, which obviously hampers maintenance. The most significant drawback, however, is the absence of modularization and a library concept; making the development and verification of bigger systems a huge burden.
In this paper, we identify several problems of KeYmaera's input syntax and illustrate them with examples. To overcome these problems, we first describe the original syntax in form of a metamodel. Then, we propose to extend this metamodel with established programming concepts such as subprogram and abrupt termination. We illustrate our extensions by using a new graphical concrete syntax. Examples from a recent KeYmaera tutorial serve for our paper as illustration examples.

Keywords: Cyber-Physical System (CPS) · Safety Property Verification · Theorem Proving · Language Design · Domain-Specific Language (DSL) · Metamodel.

1 Motivation

A Cyber-Physical System (CPS) is a system existing in the real world, which usually consists of both cyber and physical components. The behaviour of a cyber component is determined by the (computer) program, which is executed on this component while the behaviour of a physical component follows laws from physics, e.g. for torque, acceleration, velocity, etc. An important subset of CPSs are control systems consisting of sensors, processors, and actuators, whose

* The author thanks the anonymous reviewers for their detailed and helpful reviews.

correct functioning is of upmost importance and should be assured by formal verification techniques.

A hybrid system is a formal model of a CPS. To capture the behaviour of cyber components, the hybrid system needs the notion of programs. The behaviour of physical components are modelled by law in physics, which are formulated in terms of ordinary differential equations (ODEs). The theorem prover KeYmaera is able to formally verify properties of hybrid systems formulated in differential dynamic logic (DDL)[13, 18]. In this paper, we analyse DDL as used by KeYmaera as input format. We point out some obstacles of the chosen input syntax and make proposals to overcome them.

One of the main problems of the used DDL is, that this single formalism is used for three different purposes, namely, to i) describe the system to be analysed (*system description*), to ii) formulate the properties to be hold for the system (*system specification*), and to iii) formulate proofs (*system verification*). Note that a proof is a tree of DDL-formulas where each connection between nodes of the proof tree must be justified by one rule of the used proof calculus.

Thus, the very same DDL formalism serves quite different purposes and there are some cases, in which it is hard to say, which purpose a given DDL artefact actually serves. For example, the user of KeYmaera is sometimes forced to reformulate a system description in a non-intuitive way, just to make a property of this system verifiable. In other words, the property *about* the system one would like to prove has a strong influence on the way one describes the system itself! Note that - ideally - one should be able to formulate the system description fully independent from the properties one would like to prove - usually later - about the system. As we illustrate with a model of the very simple bouncing ball example, this independence is sometimes not possible. This makes the usage of KeYmaera rather an art than an engineering discipline.

The input syntax for KeYmaera is very rudimentary and forces the user to describe a system is a Big Blob, since modularization, e.g. by subsystems or subprograms, is simply syntactically not possible. In our analysis, we identify also other weaknesses, for example that the correct function of evolutional states rely on executing the right statement before entering the state or that evolutional states usually share a high portion of ODEs. Unfortunately, the current syntax makes it impossible to let an evolutional state 'inherit' from an already defined evolutional state to prevent a copy-paste style in the system description.

In addition to identifying problems of KeYmaera's input syntax, we also make proposals to overcome these problems. In order to describe our solutions at the right level of abstraction, our solution proposal will address the abstract syntax - which we define in form of a metamodel - instead of the textual concrete syntax. In order to stress the independence of our solution proposals from the concrete syntax, we will employ also a graphical syntax, which is close to the Abstract Syntax Tree (AST).

2 Background

We first review the logical basis of the prover KeYmaera.

2.1 Dynamic Logic (DL)

The term Dynamic Logic (DL) was coined for the first time by Harel et al. in [7], which is based on the work of Pratt [16] and Hoare/Floyd [4, 8]. A recent review on the history of Dynamic Logic is given by Pratt in [17].

Dynamic Logic has a long tradition in analysing *programs running on a machine*. (First-Order) Dynamic Logic allows for a program α to formulate properties for the pre- and post-state of the program's execution. Syntactically, DL formulas are built on top of arithmetic terms and arithmetic atomic formulas, such as $x < 5 + 3$. The set of DL formulas is closed under the logical junctors $\wedge, \vee, \rightarrow, \leftrightarrow$, the quantifiers $\forall$ $\exists$, and the parametrized modalities $[\alpha]$ (box), $<\alpha>$ (diamond), where α is a program. A program is syntactically defined as a tree of statements. We have assignment (:=), skip (skip), test (?) as atomic statements and nondeterministic choice ($\cup$), sequential composition (;), and iteration (*) as composed statements. Furthermore, some derived statements (as known as *syntactic sugar*) are allowed. For example, the program $if\ cond\ then\ s1\ else\ s2\ endif$ is defined as an abbreviation for $(?cond; s1) \cup (?\neg cond; s2)$. In the version of DL supported by KeYmaera, all terms (e.g. 3 + 8) including variables are of type Real, so there is no support for a sophisticated type system. For a thorough introduction to Dynamic Logic in syntax and semantics, the reader is referred to [6].

Semantically, a formula of form $\phi \rightarrow [\alpha]\psi$ claims that program α, when started in a state in which ϕ holds, might not terminate or, in case it actually terminates, will result always in a state, in which ψ holds. The second modality <> (diamond), which can occur in DL-formulas as well, has a different semantics: $<\alpha>\ \psi$ claims that program α terminates and for at least one post-state the formula ψ holds (note, that α can behave non-deterministically).

As a concrete example, let us consider the formula

$$x > 0 \rightarrow [if\ x > 0\ then\ x := x - 1\ else\ x := -25\ endif; x := x + 1]\ x > 0 \quad (1)$$

The program α within the box modality is the sequential composition (operator ;) of an if-statement and an assignment (operator :=). The claim, formulated by (1) about program α reads as follows: Whenever α is started in a state, in which $x > 0$ holds, then $x > 0$ must also hold once α has terminated (note, that termination of α is not part of the claim). Formula (1) is actually valid, i.e. under all circumstances the formula is evaluated to true (see [6] for a formal definition of validity).

It is rather easy to argue informally on the validity of (1): This implication evaluates only to false, when its premise evaluates to true and its conclusion to false. The premise is $x > 0$. Under this assumption, when executing program α,

the then-branch of the first statement (if-statement) is always taken and decreases variable x by one. In the second statement, the value of x is again increased by one, so the value of x in the post-state – let us denote it by x_{post} – is $x_{post} = x_{pre} - 1 + 1$, while x_{pre} denotes the value of variable x in the pre-state. The conclusion of (1) can thus be reduced to the proof obligation $x_{pre} - 1 + 1 > 0$, which can never evaluate to false if we assume $x_{pre} > 0$. Fortunately, we do not have to rely on informal argumentation for showing the validity of (1) but can also use the theorem prover KeYmaera, which proves (1) fully automatically.

Please note that the formulas of DL do not make any claim about the execution time of program α, but only formulate properties on the relationship of α's pre- and post-states. You might just think all statements within program α being executed instantaneously, i.e. their execution does not take any time. This is an important difference to the extension of DL, called Differential Dynamic Logic (DDL), we consider next.

2.2 Differential Dynamic Logic (DDL)

DDL [12] is an extension of DL, which means that every DL formula is also a DDL formula. The same way as a DL formula, a DDL formula usually makes a claim about a program α. However, since DDL formulas are mainly used to describe the behaviour of Cyber-Physical Systems, we rather say that program α *encodes the behaviour of the CPS* instead of α *is executed on a machine*, as we do for programs α of pure DL formulas.

The only difference between DL and DDL is a new kind of statement called *continuous evolution statement* (or simply *evolution statement*), which is allowed to occur in programs α. When during the execution of α a continuous evolution statement is reached, then the execution of this statement *takes time* and the system will stay in the corresponding *evolution state* for a while. Note that this a new semantic concept of DDL and marks an important difference to pure DL!

Executing the evolution statement means for the modelled CPS to stay in the evolution state as long as it wishes (the time to stay is - in general - chosen non-deterministically). However, the modeller has two possibilities to restrict the time period the system stays in the evolution state: The first possibility is to add a so-called *domain constraint* to the evolution statement, which is a first-order formula and which is separated from the rest of the statement by & (ampersand). The domain constraint semantically means that the system cannot stay longer in the evolution state than the time at which the constraint is evaluated to *true*. In other words: at latest when the evaluation of the domain constraint switches from *true* to *false*, the system has to leave the evolution state.

The second possibility to restrict the time period is to have a sequential composition of an evolution statement followed by a test statement. Theoretically, the machine can leave the evolution state at any time, but if the following test evaluates to *false*, then this branch of execution is dismissed for the logical analysis of the system behaviour. Thus, an evolution statement immediately followed by a test statement is a general technique to force the system to remain in the evolution state as long as the test condition is evaluated to *false*.

Bouncing Ball as a simple CPS We illustrate both the usage of an evolution statement as well as the two mentioned techniques to control the time the system will stay in the evolution state by the following bouncing ball example:

$$\alpha_{BB} \equiv (\{x' = v, v' = -g \;\&\; x \geq 0\}; ?x = 0; v := -cv)* \tag{2}$$

The behaviour of the bouncing ball is described with the help of a new kind of variables, called *continuous variables*. For example, variable x is always a non-negative number and encodes the ball's position and variable v encodes velocity, which can be both positive (going up) or negative (going down). The constant g is the gravitation acceleration and greater 0. The constant c is the damping coefficient, a number between 0 and 1.

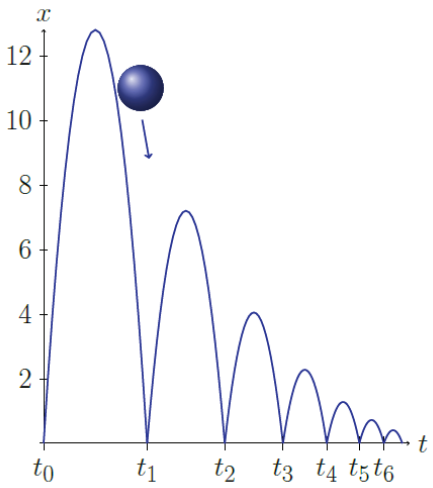


Fig. 1. Sample Trajectory of a Bouncing Ball (Source: [13, p.98])

The structure of α_{BB} is that of an iteration (operator *) over a sequence (operator ;) of an evolution statement (enclosed by the curly braces), followed by a test (operator ?), followed by an assignment (operator :=). The program α_{BB} is read as follows: The systems starts in a state with given values for variables x and v. These values are not specified yet, but later, we will force the start position x_0 to be a positive number while the start velocity v_0 is allowed to be positive, zero, or negative. As long as the system stays in the first evolution state, the values of x, v will change continuously over time according to physical laws. Thus, the continuous variables x, v represent rather functions $x(t), v(t)$ over time t. The relevant physical laws for x, v are expressed by the two differential equations: $x' = v, v' = -g$.

The latter means that the velocity decreases constantly over time due to gravitational force of the earth. Fortunately, this ODE has a simple polynomial solution, which facilitates the analysis of the whole system considerably: $v(t) = v_0 + -g * t$. Analogously, depending on the changing velocity v, the position x of the bouncing ball changes with $x(t) = x_0 + v_0 * t + \frac{-g}{2}t^2$.

The domain constraint $x \geq 0$ mentioned in the evolution statement allows the system to remain in the evolution state only as long as x is non-negative. Theoretically, the system can leave at any time the evolution state, but the next statement is the test $?x = 0$. Thus, if the system leaves the evolution state with $x > 0$, then this computational branch will be discarded. Thus, when verifying properties of the system we can rely on that the system leaves the evolution state only when $x = 0$, meaning when the ball touches the ground. The following assignment $v := -cv$ encodes that the ball goes up again: The negative value v due to the ball falling down will change instantaneously to a positive value (multiplication with $-c$) but the absolute value of v decreases since the ball loses energy when touching the ground and changing the move direction. Fig. 1 shows how the position x of a bouncing ball might change over time (sample trajectory).

3 Problems in Using KeYmaera's Input Syntax

Differential Dynamic Logic as introduced above is supported by KeYmaera and allows to verify formally important properties of technical system as demonstrated in numerous case studies from different domains, e.g. aircrafts [9, 14], trains [15], robots [11].

However, the used input syntax to formulate properties in form of DDL formulas suffers from numerous problems that are described in the following. The solutions we propose to overcome these problems are discussed in Sect. 4.

(1) Invariant specification is not directly supported in DDL Besides describing the behaviour of hybrid systems as done with program α_{BB} for the bouncing ball, the main purpose of DDL is to specify also properties of such systems. Typical and in practice very important properties are so-called *safety properties*, saying that the system never runs into a 'bad situation'. Let's encode a 'bad situation' with $\neg\psi$. We can show the absence of $\neg\psi$ by proving that in all reachable system states formula ψ holds, i.e. ψ is an invariant. If we assume all statements except the evolution state are executed instantaneously, then showing invariant ψ actually means to show that ψ holds while the system stays in any of its evolution states. However, the modality operators provided by DDL allow only to describe the state *after* the program has terminated. For example, for the bouncing ball system α_{BB} defined in (2) we can prove very easily

$$x = 0 \rightarrow [\alpha_{BB}]x = 0 \tag{3}$$

Note, however, that $x = 0$ is not proved to be an invariant! If we want to express the interesting invariant, that position x remains all the time

within the interval $[0, H]$, while H encodes the system's initial position and if velocity v is initially 0, we have to admit that the formula

$$H > 0 \wedge v = 0 \wedge x = H \wedge 0 < c \wedge c < 1 \rightarrow [\alpha_{BB}]x \leq H \tag{4}$$

is provable, but does NOT encode $x \leq H$ being an invariant because this formula does not say anything about x and H *while* the system stays in the evolution state $\{x' = v, v' = -g \ \& \ x \geq 0\}$, which is part of α_{BB}. In order prove $x \leq H$ being an invariant the user is forced to reformulate α_{BB} to

$$\alpha'_{BB} \equiv (\{x' = v, v' = -g \ \& \ x \geq 0\}; (skip \cup (?x = 0; v := -cv))* \tag{5}$$

This, however, would be an example for choosing the system description depending on the property we would like to prove, what we consider as bad style.

(2) Evolution state definition cannot be reused Evolution statements have to contain all ODEs that should hold in the corresponing states. If a program contains multiple evolution statements, then all ODEs usually have to be copied for all these statements, since an ODE normally encodes a physical law that holds in each of the evolution states. Currently, the syntax of KeYmaera does not allow to define all ODEs once and then to reuse this definition for all occurring evolution statements. This lack of reuse results in a copy-and-paste style for describing a system. As an example, we refer to Example 3a from the KeYmaera-tutorial [18], page 10, equation (20): $\{p' = v, v' = -a \ \& \ v \geq 0 \wedge p + \frac{v^2}{2B} \leq S\} \cup \{p' = v, v' = -a \ \& \ v \geq 0 \wedge p + \frac{v^2}{2B} \geq S\}$ Here, the definition of the two evolution states (in curly braces) are very similar and defined by copy-and-paste.

(3) Evolution state definition is not encapsulated In the KeYmaera-tutorials [18, 12], there is a frequently applied pattern to ensure that the system stays in an evolution state $ev \equiv \{\ldots \& \ldots\}$ for at most time ϵ. This is achieved by extending the definition of ev to $ev' \equiv \{\ldots, t' = 1 \& \ldots \wedge t \leq \epsilon\}$ while t is a fresh continuous variable. Together with the ODE $t' = 1$, the additional domain constraint $t \leq \epsilon$ forces the system to leave ev' at latest after time ϵ has elapsed. However, this refined definition of ev works only, if the value of t has been set beforehand to 0. In order to achieve this, the statement ev is usually substituted by $t := 0; ev'$. While this pattern works basically in practice, the definition of ev' is not encapsulated and prevents compositionality of programs.

(4) Missing notion of subprogram (or function call in general) Once the examples in the KeYmaera-tutorials [18, 12] become a little bit more complicate, they are given in a composed form, e.g. Example 3a from [18, p.10]: $init \rightarrow [(ctrl; plant)*]req$ where $init \equiv \ldots$, $ctrl \equiv \ldots$, $plant \equiv \ldots$, $req \equiv \ldots$ Presenting a DDL problem in such a composed form highly improves readability. However, the usage of such a composed notation is impossible for the input file of KeYmaera. While one could imagine to introduce new relational symbols $init, req$ and to constrain their interpretation by subformulas $init \leftrightarrow \ldots$, $req \leftrightarrow \ldots$, it is currently impossible to define subprograms $ctrl$ and $plant$ and to compose the resulting program from these subprograms.

4 A Metamodel-based Approach to Solve Identified Problems

The problems identified above can be overcome by incorporating language concepts from object-oriented programming languages and statecharts into the input syntax of KeYmaera. In order to discuss the incorporated new language concepts at the right level of abstraction, we formulate our proposal in form of a changed metamodel for KeYmaera's input syntax. As a starting point, we present the metamodel of the current syntax.

4.1 Metamodel of Current KeYmaera Syntax

Metamodeling [5] is a widely adopted technique to specify the abstract syntax of modelling and programming languages. One well-known language definition is that of the Unified Modeling Language (UML) [19].

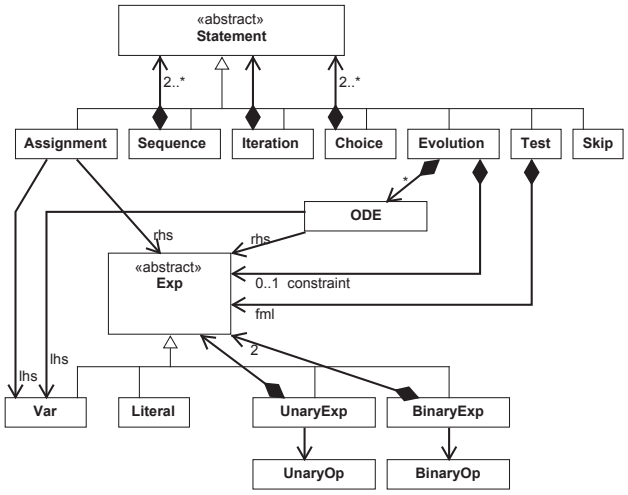


Fig. 2. Metamodel of KeYmaera's Input Syntax (Part of Statement)

Fig. 2 shows a sketch of the metamodel of KeYmaera's current input syntax with focus on statements within a program. All metaassociations with multiplicity greater than 1 are assumed to be ordered. If the multiplicity on a metaassociation is missing, then 1 is the default value. The metaclass `Exp` represents expressions of both type *Real* (e.g. 5 + x) and of type *Boolean* (e.g. x < 10).

A concrete program α for KeYmaera can be represented by an instance of the metamodel. This instance is equivalent to the result obtained by parsing this program, i.e. the abstract syntax tree (AST).

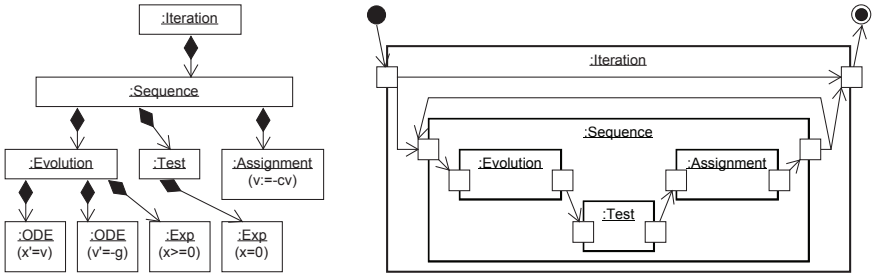


Fig. 3. Instance of the Metamodel(left) and Control-Flow Inspired Graphical Syntax (right) for Bouncing Ball Program (α_{BB})

The left part of Fig. 3 shows a sketch of the metamodel instance for the bouncing ball program $\alpha_{BB} \equiv (\{x' = v, v' = -g \;\&\; x \geq 0\}; ?x = 0; v := -cv)*$ as defined in (2). In the right part we see an AST-aligned graphical representation of the same program: Each kind of statement is represented by a block with input and output pins. The control flow is visualized by directed edges connecting two pins. The pre-/post-states of the program execution are represented by the symbol for start/final state known from UML's statemachine [19].

Based of this graphical notation we discuss now solutions for the problems listed in Sect. 3.

4.2 Solutions for Identified Problems

(1) Invariant specification is not directly supported in DDL

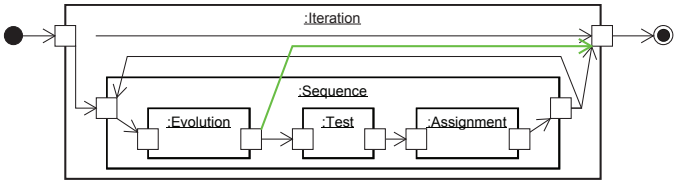


Fig. 4. Solution for Invariant Specification Problem

As described in Sect. 3, the modal operator $[\alpha]$ refers always to the post-state represented by the final state node in Fig. 3, right part. However, for checking an invariant we need a reference to the state after each Evolution-statement has been finished. This moment in the execution is represented by the output-pin of the evolution state. What is needed in the program

semantics is a direct edge from each output-pin of each evolution state to the final state, as shown in Fig. 4 by the green edge. This concept is known as *abrupt termination*.

Note that abrupt termination could be realized without any change of the input syntax of KeYmaera since it requires merely a changed control-flow for the existing statements.

(2) Evolution state definition cannot be reused When within multiple evolution statements repeat again and again the same ODEs and constraints, the readability of the program suffers. To prevent this, our proposal is to introduce the *declaration of* named evolution statements which can be referenced by other evolution statement to - for example - inherit from them ODEs and constraints. The relevant change of the metamodel is shown in Fig. 5.

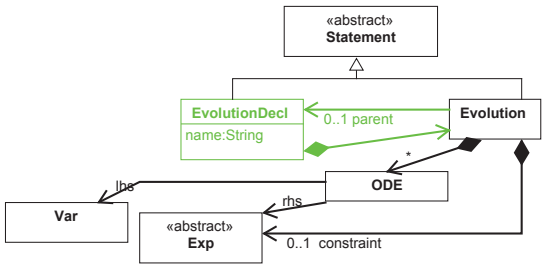


Fig. 5. Solution for Evolution State Reuse Problem

One problem still to be discussed is, whether the declaration of an evolution state can occur at an arbitrary location in the program or should be rather done prior to the program as a global declaration. This question refers to the important issue of which scope the identifier introduced by the declaration (see metaattribute `name`) should actually have. Since resolving the scope of an identifier is rather a problem when parsing a program, this issue is out of scope for this paper.

(3) Evolution state definition is not encapsulated As demonstrated in the problem definition, an evolution statement sometimes works only as intended, when a variable has been set beforehand to the right value. Practically this means, the evolution state EV is always prepended by an assignment $ASGN$, so $(ASGN\ ;\ EV)$ has to occur always for correctness. In order to get rid of dependencies of evolution state to assignments from the context (which prevents a simple reuse of EV within a different context), we propose to extend the evolution state with optional additional statements that are always executed when entering or leaving the state. This state ex-

tension is well-known as entry-/exit-actions from UML statemachines. The relevant change of the metamodel is shown in Fig. 6.

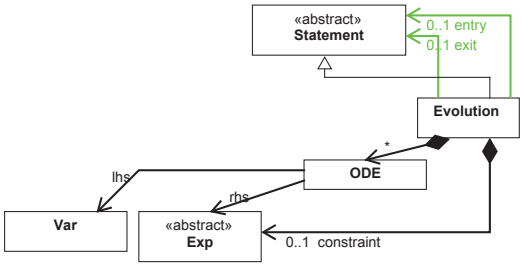


Fig. 6. Solution for Evolution State Encapsulation Problem

(4) Missing notion of subprogram One of the most basic concepts in programming is the possibility to encapsulate (a block of) statements with a given name and to reuse these statements at various locations of the program. This concept is usually called *subprogram*, *procedure*, or *method*; depending whether parameters are used or not. In general, this is a very old, proven and well understood concept, so that we introduce only the most simple variant in our solution proposal here (cmp. Fig. 7).

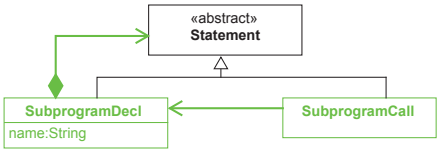


Fig. 7. Solution for Missing Subprogram Problem

5 Towards the Realization of Solution Proposals

In this section we review possible realization options for the proposed solution. Finally, we give a recommendation for one realization option, but the realization itself is currently work in progress and will be described elsewhere.

5.1 Realization by Extending the Prover KeYmaera

The prover KeYmaera mainly consists of a parser for the input syntax and a calculus in form of proof rules, which even can be changed by the user. In addition, there are some technical components such as (i) a prover engine for applying proof rules to create a formal proof, (ii) adapters to incorporate external proof systems such as *Z3* or *Mathematica*, and (iii) a GUI to control the proof editing process. However, all these technical components are out of scope for this paper.

For our proposals it is worth to distinguish pure syntactic changes from those, that have an impact on the calculus used by KeYmaera. To the latter belong the support of abrupt termination (problem (1)) and the possibility to invoke subprograms (problem (4)). These changes would require to considerably extend KeYmaera's calculus. While such an extension requires intimate knowledge of the underlying proof engine, it is nevertheless possible, as the KeY-system [1][1] demonstrates. The KeY-System is an interactive verification tool for programs implemented in the language Java and its calculus covers all the subtleties of a real world programming language, including *function calls*, *call stack*, *variable scope*, *abrupt termination* by throwing an exception, *heap analysis*, etc.

Pure syntactic changes among our proposals, i.e. addressing problems (2), (3), could be realized in KeYmaera just by extending the parser. Note that the creation of an alternative concrete input syntax is also topic of the ongoing project called Sphinx [10] carried out by the authors of KeYmaera. Sphinx aims to add a graphical frontend to the prover and will allow the user to specify a program in a pure graphical syntax (similar to our graphical notation proposed in Fig. 3, right part).

The general problem with any deep change of the prover KeYmaera is the technical knowledge it requires. Furthermore, there are good reasons to keep a version of the tool with the original syntax due to its simplicity, what makes it much simpler to use KeYmaera for teaching than, for example, its predecessor KeY. However, a new version of KeYmaera with deep changes is hard to maintain as the original KeYmaera might evolve in future. For these reasons, deep changes can hardly be done by others than the original authors of KeYmaera themselves.

5.2 Realization by Creating a Frontend-DSL

An alternative and flexible approach is the development of a frontend-DSL to incorporate the new language concepts introduced in Sect. 4.2. The main idea is to develop a new Domain-Specific Language according to the given metamodel. Note that the metamodel covers merely the abstract syntax and keeps some flexibility for the concrete syntax. Modern frameworks for defining DSLs such as Xtext and Sirius even allow to have for one DSL

[1] Historically, the KeY-system is the predecessor of KeYmaera.

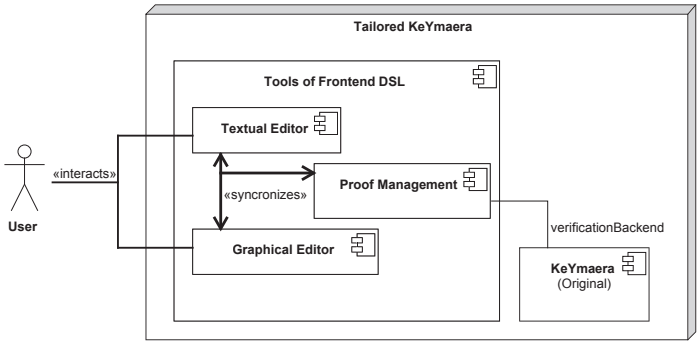


Fig. 8. Architecture of Solution using Frontend-DSL

multiple representations (i.e. *concrete syntaxes*) supported by corresponding editors, e.g. a textual syntax and a graphical syntax. Fig. 8 shows the general architecture of such a tool.
Note that the new tool will allow the user to interact synchronously with both a textual and a graphical editor to create a model. However, the new models cannot be simply transformed to input files for the original KeYmaera, because the new syntax supports some semantically new concepts such as abrupt termination or subprogram invocation stack. It is the task of the ProofManagement component to split tasks - for instance to prove an invariant - into smaller proof obligations, which can be formulated as formulas of differential dynamic logic (DDL) and to pass these obligations to the original KeYmaera tool as verification backend. How an invariant task can be split into smaller proof obligations is demonstrated based on a concrete example in [2].

6 Related Work

The definition of DSLs can be done with numerous technologies, e.g. Xtext, Spoofax, Metaedit, MPS. For realizing a DSL with both a textual and a graphical concrete syntax, the combination Xtext and Sirius is very attractive.
Enriching the prover KeYmaera with a graphical syntax for DDL programs is done in the project Sphinx [10]. The architecture of this tool is pretty similar to our proposal in Fig. 8, but the focus is - in difference to our approach - not the improvement of readability and modularization by making the input syntax richer, but to enable the user to graphically construct a program for DDL.
While enriching a plain, imperative language with concepts from object-orientation has been done many times in computing science's history (take

the transition from C to C++ or from Modula to Oberon as examples), it is still considered as a challenge. There is an excellent tutorial by Bettini in [3] on how to incorporate into a plain sequential language based on simple expressions additional concepts from object-oriented programming (e.g. *class*, *attribute*, *method*, *visibility*). The resulting language in this tutorial is called *SmallJava* and illustrates almost all technical difficulties when realizing a Java-like programming language in form of a DSL.

7 Conclusion and Future Work

The syntax of programs of differential dynamic logic as supported by the theorem prover KeYmaera have been kept very simple and low level. An advantage of this decision is that also the calculus for proving such programs being correct could be kept relatively simple and that proofs can be constructed and understood easily. At the downside we have that - once the examples become a little bit more complicate - programs are hard to read, poorly structured, and are impossible to reuse within a different context.
In this paper, we identified four general problems when applying the current program syntax in practice. Furthermore, we made proposals to overcome the identified problems by incorporating proven language concepts from programming languages and from UML's statemachines into KeYmaera's input syntax. These concepts have the potential to make programs scalable and easier to be understood since they foster readability and modularization.
Our proposals have been formulated in form of a changed metamodel representing the abstract syntax of programs. The chosen form for formulating the proposal has the advantage of being very precise while leaving it open, how the changes should actually be realized in a given concrete syntax. Currently, the implementation of a frontend DSL being the main constituent of a *Tailored KeYmaera* tool set is under construction, but not finished yet.

References

1. Ahrendt, W., Beckert, B., Bubel, R., Hähnle, R., Schmitt, P.H., Ulbrich, M. (eds.): Deductive Software Verification - The KeY Book - From Theory to Practice, Lecture Notes in Computer Science, vol. 10001. Springer (2016)
2. Baar, T., Staroletov, S.: A control flow graph based approach to make the verification of cyber-physical systems using KeYmaera easier. Modeling and Analysis of Information Systems. 2018;25(5) pp. 465–480 (2018)
3. Bettini, L.: Implementing Domain-Specific Languages with Xtext and Xtend. Packt Publisher, 2nd edn. (2016)
4. Floyd, R.W.: Assigning meanings to programs. In: Schwartz, J.T. (ed.) Proceedings of Symposium on Applied Mathematics. pp. 19–32. Mathematical Aspects of Computer Science, American Mathematical Society
5. Gonzalez-Perez, C., Henderson-Sellers, B.: Metamodelling for software engineering. Wiley (2008)

6. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. Foundation of Computing, MIT Press (2000)
7. Harel, D., Meyer, A.R., Pratt, V.R.: Computability and completeness in logics of programs (preliminary report). In: Hopcroft, J.E., Friedman, E.P., Harrison, M.A. (eds.) Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA. pp. 261–268. ACM (1977)
8. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–580 (1969)
9. Jeannin, J., Ghorbal, K., Kouskoulas, Y., Gardner, R., Schmidt, A., Zawadzki, E., Platzer, A.: A formally verified hybrid system for the next-generation airborne collision avoidance system. In: Baier, C., Tinelli, C. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015. LNCS, vol. 9035, pp. 21–36. Springer (2015)
10. Mitsch, S.: Modeling and Analyzing Hybrid Systems with Sphinx – A User Manual. Carnegie Mellon University and Johannes Kepler University (2013), available from: http://www.cs.cmu.edu/afs/cs/Web/People/smitsch/pdf/userdoc.pdf
11. Mitsch, S., Ghorbal, K., Platzer, A.: On provably safe obstacle avoidance for autonomous robotic ground vehicles. In: Newman, P., Fox, D., Hsu, D. (eds.) Robotics: Science and Systems IX, Technische Universität Berlin, Berlin, Germany, June 24 - June 28, 2013 (2013)
12. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer, Heidelberg (2010)
13. Platzer, A.: Logical Foundations of Cyber-Physical Systems. Springer (2018)
14. Platzer, A., Clarke, E.M.: Formal verification of curved flight collision avoidance maneuvers: A case study. In: Cavalcanti, A., Dams, D. (eds.) FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5850, pp. 547–562. Springer (2009)
15. Platzer, A., Quesel, J.: European train control system: A case study in formal verification. In: Breitman, K.K., Cavalcanti, A. (eds.) Formal Methods and Software Engineering, 11th International Conference on Formal Engineering Methods, ICFEM 2009, Rio de Janeiro, Brazil, December 9-12, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5885, pp. 246–265. Springer (2009)
16. Pratt, V.R.: Semantical considerations on floyd-hoare logic. In: 17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-27 October 1976. pp. 109–121. IEEE Computer Society (1976)
17. Pratt, V.R.: Dynamic logic: A personal perspective. In: Madeira, A., Benevides, M.R.F. (eds.) Dynamic Logic. New Trends and Applications - First International Workshop, DALI 2017, Brasilia, Brazil, September 23-24, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10669, pp. 153–170. Springer (2017)
18. Quesel, J.D., Mitsch, S., Loos, S., Aréchiga, N., Platzer, A.: How to model and prove hybrid systems with KeYmaera: A tutorial on safety. STTT **18**(1), 67–91 (2016)
19. Rumbaugh, J.E., Jacobson, I., Booch, G.: The unified modeling language reference manuel - covers UML 2.0, Second Edition. Addison Wesley object technology series, Addison-Wesley (2005)

Data Compression Algorithms in Analysis of UI Layouts Visual Complexity

Maxim Bakaev [0000-0002-1889-0692], Ekaterina Goltsova, Vladimir Khvorostov, Olga Razumnikova [0000-0002-7831-9404]

Novosibirsk State Technical University, Novosibirsk, Russia
bakaev;xvorostov;razumnikova@corp.nstu.ru

Abstract. Measuring visual complexity (VC) of human-computer user interfaces (UIs) sees increasing development, as VC has been found to affect users' cognitive load, aesthetical impressions and overall performance. Spatial allocation and ordering of UI elements is the major feature manipulated by an interface designer, and in our paper we focus on perceived complexity of layouts. Algorithmic Information Theory has justified the use of data compression algorithms for generating metrics of VC as lengths of coded representations, so we consider two established algorithms: RLE and Deflate. First, we propose the method for obtaining coded representations of UI layouts based on decreasing of visual fidelity that roughly corresponds to the "squint test" widely used in practice. To confirm applicability of the method and the predictive power of the compression algorithms, we ran two experimental surveys with over 4700 layout configurations, 21 real websites, and 149 participants overall. We found that the compression algorithms' metrics were significant in VC models, but the classical purely informational Hick's law metric was even more influential. Unexpectedly, algorithms with higher compression ratios that presumably come closer to the "real" Kolmogorov complexity did not explain layouts' VC perception better. The proposed UI coding method and the analysis of the compression algorithms' metrics can contribute to user behavior modeling in HCI and static testing of software UIs.

Keywords: Algorithmic Complexity, Static UI Analysis, Human-Computer Interaction, Information Processing.

1 Introduction

1.1 Visual Complexity in Human-Computer Interaction

A few decades ago, with the increasing ubiquity of computers and the growing number of users, visual complexity (VC) started becoming a research field of its own, detaching itself from the general studies of complex systems [1]. Nowadays it is well-known in human-computer interaction (HCI) that perceived user interface VC significantly affects not just cognitive load, but also user preferences, aesthetical and other affective impressions ([2]-[5]). The general guideline in HCI is that all other things being equal, VC should be decreased (in some cases, a certain level of complexity should be maintained, due to reasons

of aesthetic perception). However, we so far lack universally accepted quantitative measure and the respective techniques for automated assessment of user interface VC, although their development is largely seen as desirable [6].

One of the obstacles in tackling this problem is that VC is not universal and the factors and features affecting it depend of the object being perceived. For instance, for certain signs (hieroglyphs) these factors included area and, correspondingly, the number of lines and strokes, while for shapes of familiar objects it was the number of turns [7]. Most current research works seem to focus on images (even more often, on photos), while publications related to complexity in data visualization and user interfaces (UIs) are relatively scarce. As some examples, we can note [8] and [9], where the authors proposed the formulas and developed software tools for calculating the UI complexity values, as well as [10].

1.2 Complexity and Data Compression

Still, there are universal approaches in VC research and quantification, and they are based on Shannon's Information Theory and on Gestalt principles of perception [11]. The former provides robust quantitative apparatus for measuring information content (calculating entropy), but it has been repeatedly shown that information-theoretic complexity does not correspond to human visual perception well, particularly since it does not consider spatial structures [12]. The latter has the concept of "visual simplicity" as the foundational principle, and has been shown to match the humans' "top-down" perception of objects well. However, it used to suffer from lack of quantitative methods, at least until the emergence of Algorithmic Information Theory (AIT), which linked this approach to Kolmogorov algorithmic complexity [13].

With AIT it became possible to directly link the concepts of "simplicity" and "probability" and unite the two corresponding approaches. The complexity of a percept is the length of the string that generates the percept and at the same time expressed through Kolmogorov probability of the percept. The compression algorithm acts as a practical substitute of the universal Turing machine, taking the compressed string (previously produced with the same algorithm) to reproduce the original string. Kolmogorov complexity is defined as the length of the shortest program needed to produce a string – hence, the length of the compressed string can stand for the pseudo-Kolmogorov complexity of the original string [11]. Higher compression ratios presumably allow the compressed string's length to approach the "real" Kolmogorov complexity.

1.3 Related Work and Research Question

The compression algorithm probably seeing the widest use for producing VC is JPEG (as specified e.g. in ISO/IEC 10918-1:1994 standard), which was specifically designed to consider particulars of images perception by humans. Some alternatives include calculating Subband Entropy [14] or Fractal Dimension of the image, using Zipf's Law, preliminary edge detection filters application, etc. [15].

In HCI and UI analysis, the above approaches have been successfully applied to images (JPEG compression algorithm, frequency-based entropy measures, image

types [15], etc.), textual content (characters recognition in fonts, graphic complexity [16], etc.), high-order UI design measures (e.g. amount of whitespace in [2]) and so on. Meanwhile, UI layouts have not been in the focus of quantitative VC research, even though they are known to have significant impact on users' perception of UIs and are important for preserving their attained experience with a computer system [17]. To the best of our knowledge, [18] was the only work to propose a layout complexity metric, based on spatial allocation and diversity of UI elements, but this research direction seemingly failed to gain momentum.

In our work we focus on UI layouts, exploring if humans' perception of their complexity can be well explained with the measures supplied by data compression algorithms. In Methods, we briefly reiterate on the Hick's law, acting as the baseline information-theoretic measure, and describe the compression algorithms used in our study: RLE-based one and classical Deflate. We further introduce our method for coding the considered visual objects – UI layouts. Since cases were reported when vertical or horizontal alignment of the same UI elements mattered in terms of visual search time [19], we also investigate the different ways to convert two-dimensional layouts into bit string representations. In Section 3, we describe experimental research we performed with the model layouts and analyze the data we obtained from 78 participants. In Section 4, we verify our findings with real UIs of 21 operating websites, assessed by 63 subjective evaluators and coded according to the proposed method. In Conclusions, we summarize our results, note limitations of our study and outline prospects for further research.

2 Methods

Almost immediately after its emergence, the Shannon's Information Theory was applied to psychological and perception problems. The prerequisite for using information concepts in visual perception was measuring the information content of stimuli [11]. Arguably the most influential undertaking with regard to the cognitive aspect was the one by W.E. Hick (1952), who postulated that reaction time (RT) when choosing from equally probable alternatives is proportional to the logarithm of their number (N_H):

$$RT \sim \log_2(N_H + 1) \tag{1}$$

Despite the demonstrated applicability of the Hick's law for certain aspects related to UI design, it currently has little use in HCI [20]. First, calculating the informational content of stimuli in practice is generally more problematic. Second, it is believed that since the information-theoretic approach is analytical by nature, it is fundamentally limited in explaining human perception, which is mostly top-down: that is, focused on higher-order images and structures. As we mentioned before, AIT made it possible to bond Information Theory with Gestalt principles of perception, which are concerned exactly with how visual sensory input is organized into a percept. Within this school, it was empirically shown that the coded string lengths, the complexity measures, and performance measures in experiments are highly correlated [11]. Thus it is natural to assume that compressibility and complexity are linked in UIs as well.

2.1 The Compression Algorithms

Run-Length Encoding (RLE). RLE is one of the oldest and simplest algorithms for lossless data compression, which was already in use for handling images in the 1960s. The idea of the algorithm is relatively straightforward: *runs* of data are replaced with a single data value and the count of how many times it's repeated. Runs are sequences in which the same data value occurs several times, and these are quite common in icons, line drawings and other imagery with large mono-colored areas. The algorithm doesn't deal with 2D images, but can work with the linear string of bytes containing serialized rows of the image. In the best case, a string of some 64 repeating value would be compressed into 2 bits, i.e. providing *compression ratio* of 32 or *data rate saving* of 0.96875. It should be noted that for some kinds of files, such as high-quality photographic images, the RLE algorithm compression may even increase the volume, due to lack of runs. Based on RLE encoding principle, a number of more sophisticated algorithms and compressed data file formats were developed (.TGA, .PCX, etc.), but in our work we are going to employ a simple and straightforward RLE implementation (see in Appendix A).

Deflate. Deflate is a lossless data compression algorithm that is based on combination of *LZ77* algorithm and *Huffman coding*. It was developed by P. Katz, who used it in PKZIP archiving software, and was later defined in RFC 1951 specification. It is free from patent protections, so many compressed data formats (e.g. .PNG images) rely on Deflate. Today's popular implementation of the algorithm is in widely used *zlib* software library, which is also capable of compressing data according to the somewhat adjusted RFC 1950 and RFC 1952 *(gzip)* variations.

In the first stage of Deflate, LZ77-based "sliding window" approach is used to replace duplicate series of data (strings) with back-references to a single copy of that data. In the second stage, commonly used symbols are replaced with shorted representations and less common symbols – with longer ones, based on Huffman coding method. The Huffman algorithm (which doesn't guarantee optimal result, but has very reasonable time complexity) produces a variable-length code table for encoding source symbols. The Huffman codes are "prefix-free", i.e. an encoding bit string is never a prefix for another encoding bit string. The Huffman coding is now widespread, being used in compression of many kinds of data, especially photos (JPEG) and multimedia, as well as in data transmission protocols. At the same time, it does provide good data compression ratio for small alphabet sizes.

For the purposes of current research work, we relied on PHP's standard *zlib_encode* function, called with ZLIB_ENCODING_RAW parameter (corresponding to the RFC 1951 specification), to obtain the compressed string.

2.2 The "Squint" Coarsening Method

To turn layouts (as visual objects) into string representation, in our study we propose a novel method, which is based on decrease of UI visual fidelity (coarsening) – overlaying 2D grid. Layout grids with blocks aligned vertically and/or horizontally, are

de-facto standard in modern interface design, and they are implemented in Bootstrap, Axure and many other tools. Each cells of the overlaid grid contain either 1 (the area has interface elements) or 0 (few or no perceived interface elements). Such uniformity allows focusing on layout and eliminating other factors, such as perception of colors, diversity of elements, etc. Naturally, it makes the method inadequate for UI studies that involve user tasks and actual interactions, textual content, etc.

At the same time, the aforementioned coarse model still reflects most layout-related aspects of real UIs: visual organization (hierarchy), elements' weights and forms, edges, whitespace, etc. Perception-wise, Gestalt principles (particularly proximity and continuation) remain legitimate, while most UI grid quality/layout metrics can be calculated on the model: balance, symmetry, alignment points, etc. The grid model also supports scanning (thus matching the users' prevalent way of interacting with new web pages), during which the quick but persistent impressions of the UI are known to be made.

Overall, the method can be said to correspond to the informal "squint test" popular in practical UI design, which allows estimating the quality of interface elements' visual organization (see software implementation of the coarsening e.g. in [21]). The approach also starts seeing methodological use in research – for instance, in [22] they used similar method in studying user attention distribution in interaction with 2D graphic UIs.

2.3 Hypotheses and the Experimental Material

Based on the research objectives and the related work, we formulated the following hypotheses for our experimental investigation:

1. Lengths of strings output by the data compression algorithms should explain layouts' complexity perception in humans better than the baseline measure. For the latter we are using the purely informational Hick's law, lacking the spatial allocation consideration.
2. The algorithm providing higher data compression ratio better explains the layout complexity perception, since its output is closer to the algorithmic complexity.
3. The conversion of two-dimensional layout into bit string sent to a compression algorithm is uniform for vertical and horizontal dimensions (we consider only these 2 types of the filling curve) – i.e. it does not consistently affect the measure's explanatory power.

Layouts (Experiment 1). For the model testing of the hypotheses and assessment of the coarsening method's applicability, we used two-dimensional grids (all square, to better align with the hypothesis #3). In the grid, square cells of the same sizes were allocated, most of which were white, corresponding to zeros, while a varying number of them were filled with blue color, corresponding to ones. In Fig. 1 we show example of the grid, with the corresponding numerical values overlaying the cells (in the experiment they did not show to participants). The two-dimension matrix corresponding to the example is: [[0,0,1,0,0],[1,0,0,0,1],[0,0,1,0,0],[0,1,0,1,1],[0,1,0,0,0]].

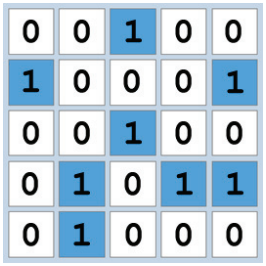


Fig. 1. Example of the grid with explanation of the numerical values corresponding to the cells.

Websites (Experiment 2). The goals were to check if our model results generalize to real UIs and if the coarsening method can be feasible in practice. Since there are many interfering factors, not just layouts, we should expect to find smaller statistical effects (this is why our first experiment was with models). We chose to focus on web interfaces, so that the popular JPEG algorithm could be more dependable in producing the additional baseline measure. So, we employed 21 operating websites of 11 German universities and 10 Russian ones – in all cases, English versions were used. The websites for the experiment were manually selected, with the requirements that 1) the universities are not too well-known, so that their reputations do not bias the evaluations; 2) the designs are sufficiently diverse in terms of layout, colors, images, etc.

3 Experiment 1: Layouts

3.1 Experiment Description

Participants. The overall number of valid subjects in our experimental sessions undertaken within one month was 78 (57 females, 20 males, 1 undefined). Most of them were Bachelor and Master students of Novosibirsk State Technical University, who took part in the experiment voluntary (no random selection was performed). The subjects' age ranged from 17 to 65, mean 21.7 (SD = 2.03). All the participants had normal or corrected to normal vision and reasonable experience in IT usage. Most of the participants worked from desktop computers installed in the university computer rooms. There were also 17 other registrations with the online surveying system, but none of those subjects completed the assignment (on average, each of them completed only 4.4% of the assigned evaluations), so they were discarded from the experiment.

Design. The experiment used within-subjects design. The main independent variables were:

- Number of filled cells in the grid (the "ones" in the matrix), ranging from 4 to 13: N;
- Number of all cells in the grid (elements in the matrix). We used two levels, 25 (5*5 grid) and 36 (6*6 grid): S_0;

- Layout configuration – i.e. allocation of filled cells in the grid, which was performed randomly.

For the purposes of the compression algorithms application, the layout configurations needed to be converted into bit string. We used two ways to do that: by rows (matrix [[1,1],[0,0]] becomes [1100] string) and by columns (the same matrix becomes [1010]), in both cases starting from the top left element. So, we also got several "derived" independent variables in the experiment:

- Lengths of the row- and column-based bit strings compressed with the RLE-based algorithm: $L_{RLE\text{-}R}$ and $L_{RLE\text{-}C}$;
- Lengths of the row- and column-based bit strings compressed with the Deflate algorithm: $L_{D\text{-}R}$ and $L_{D\text{-}C}$;
- The corresponding data compression ratios for the algorithms: $C_{RLE\text{-}R}$, $C_{RLE\text{-}C}$, $C_{D\text{-}R}$, $C_{D\text{-}C}$.

The dependent variable was Complexity – the subjects' subjective evaluations of presented layouts complexity, ranging from 1 (lowest complexity) to 5 (highest complexity).

Procedure. To support the experimental procedure, we used our specially developed web-based software. Before the experiment, we used it to collect data on the participants (gender, age, university major, etc.). In each trial, the subject was shown a layout with random allocation of the colored cells (configuration) and varying N and S_0 and asked to evaluate it per the Complexity scale. The values of all the variables would be saved be the software, and the participant moved to the next trial. The configurations were independent between the trials, and the overall number of layouts to be evaluated by each subject was set to 100. The interface of the software could be either in Russian or in English, with the scale also dubbed in German.

3.2 Descriptive Statistics

The total number of layouts used in the experiment was 4734. The subjects submitted 7800 evaluations in total, and averaged evaluations for 4702 layouts (99.3%) were considered valid. Save for outliers, completing each trial on average took a participant 13.69 s, so the average time spent on an experimental session was 22.82 minutes.

The averaged value for Complexity in the experiment was 2.58 (SD = 0.96)[1]. The Pearson correlation between S_0 and N was significant, but quite low ($r = 0.096$, $p < 0.001$).

To test the hypothesis #3 (differences for row- and column-based strings compression) we used t-tests, which found no statistically significant differences for $L_{RLE\text{-}R}$ and $L_{RLE\text{-}C}$ ($t_{4701} = -0.808$, $p = 0.419$, $r = 0.708$) or $L_{D\text{-}R}$ and $L_{D\text{-}C}$ ($t_{4701} = -1.035$,

[1] We are aware about the controversy existing in the research community about treating Likert and other ordinal scales as rational ones for some methods. In our analysis we tried to use methods appropriate for ordinal scales when possible, but nevertheless were not restricted to them, if more robust analysis could be performed. We ask the readers to judge for themselves whether the potential bias in the results overweighs their usefulness.

p = 0.301, r = 0.590). So, in the further analysis we used the "best" values, equal to the minimal length or maximal compression ratio for each string:

$$L_{RLE} = \min\{L_{RLE-R}; L_{RLE-C}\} \quad (2)$$

$$L_D = \min\{L_{D-R}; L_{D-C}\} \quad (3)$$

$$C_{RLE} = \max\{C_{RLE-R}; C_{RLE-C}\} \quad (4)$$

$$C_D = \max\{C_{D-R}; C_{D-C}\} \quad (5)$$

Ranges, means and standard deviations for the considered independent variables are shown in Table 1. We found statistically significant Pearson's correlations for L_{RLE} with L_D ($r = 0.718$, $p < 0.001$), $\log_2 N$ with L_{RLE} ($r = 0.736$, $p < 0.001$), and $\log_2 N$ with L_D ($r = 0.702$, $p < 0.001$). At the same time, t-test suggested statistically significant difference between C_{RLE} and C_D ($t_{4701} = -87.5$, $p < 0.001$, $r = 0.734$). Since the compression ratios for the two algorithms in the experiment were different, the testing of our hypothesis #2 would be possible.

3.3 Effects of Independent Variables

In Table 1 we also show Pearson correlations between Complexity and the respective independent variables, all of which were significant ($p < 0.001$).

Table 1. Descriptive statistics and the correlations for the independent variables (experiment 1).

Variable	Range	Mean (SD)	r (Complexity)
S_0	25; 36	-	0.163
N	4-13	7.46 (2.01)	0.535
$\log_2 N$	2.00-3.70	2.84 (0.42)	**0.539**
L_{RLE}	8-30	17.14 (3.54)	0.531
L_D	8-20	13.95 (1.99)	0.440
C_{RLE}	1.09-4.50	1.85 (0.40)	-0.400
C_D	1.39-4.00	2.23 (0.42)	-0.199

With respect to the hypothesis #1, we can note that the strongest correlation ($r = 0.539$) was found for $\log_2 N$, which suggests Hick's law is rather applicable for explaining the perception of layouts' complexity. However, the differences in correlations are marginal compared to the ones found for N and L_{RLE}, which implies the need for further analysis.

With respect to the hypothesis #2, while Deflate algorithm provided significantly better compression ratio (mean of 2.23 vs. 1.85 for RLE), thus finding more regularities in the layouts. However, its correlation with the perceived complexity was notably weaker than for RLE algorithm. We will further elaborate on this in the regression analysis.

3.4 Regression Analysis

To explore whether the variables considered in the paper could explain layout complexity perception, we performed regression analysis for Complexity with two groups of factors. The informational component was $\log_2 N$, as having the highest correlation with Complexity and being best theoretically justified by the Hick's law. The regression model with this single factor was significant ($F_{1,4700} = 1926$, $p < 0.001$), but had relatively low $R^2 = 0.291$:

$$Complexity = -0.95 + 1.24 \log_2 N . \quad (6)$$

Further, we attempted regression with all 7 independent variables (see in Table 1) as factors. We used Backwards variable selection method, which led to the model with just two significant factors: $\log_2 N$ (Beta = 0.323, $p < 0.001$) and L_{RLE} (Beta = 0.294, $p < 0.001$). With respect to the hypothesis #2, we should specially note that the L_D factor was not significant ($p = 0.552$). The model had somehow improved $R^2 = 0.330$ ($F_{2,4699} = 1158$, $R^2_{adj} = 0.330$):

$$Complexity = -0.9 + 0.74 \log_2 N + 0.08 L_{RLE} . \quad (7)$$

To evaluate the quality of the two models that had different number of factors (k), we also calculated Akaike Information Criterion (AIC) using the following formulation for the linear regression:

$$AIC = 2k + n \ln(RSS) , \quad (8)$$

where n is sample size (in our case, n = 4702), RSS are the respective residual sums of squares. AIC for (6) amounted to 37758, while AIC for (7) was 37490, which suggests that the "information loss" of the second model is lower and it should be preferred over the first one.

4 Experiment 2: Websites

The first stage was experimental survey in which we collected subjective evaluations for website homepages, per dimensions related to visual complexity (see [23] for more detail). In the second stage, the web UIs were processed by annotators to be converted into coded representations.

4.1 Experiment Description (Subjective Complexity)

Participants. In total, 63 participants (30 male, 33 female) provided their evaluations of the websites' complexity. The convenience sampling method was applied, with most of the participants being students or universities staff members. The self-denoted age ranged from 19 to 72, mean 27.6, SD = 8.07. The self-denoted nationalities were Russian (65.1%), German (17.5%), Argentinian (4.8%), and others (including Bulgar-

ian, Vietnamese, Korean, etc.). Submissions by another 13 participants were discarded as being incomplete (none of them had at least 50% of websites evaluated).

Design. Since providing the evaluations in absolute numbers would be unattainable for the participants who were not web design professionals, we chose to rely on ordinal values. For each of the following statements, 7-point Likert scale was used (1 being "completely disagree", 7 – "completely agree"), resulting in the respective ordinal variables:

- "This webpage has many elements." SElements
- "The elements in the webpage are very diverse." SVocab
- "The elements in the webpage are well-ordered." SOrder
- "The webpage has a lot of text." SText
- "The webpage has a lot of graphics." SImg
- "The webpage has a lot of whitespace." SWhite
- "The webpage appears very complex." SComplex

In the current work, we only used SComplex as the dependent variable in the experiment 2. We also employed SElements and SWhite to extra check the applicability of the proposed coarsening method implementation, which we describe further.

Since for web UIs were are able to use the JPEG algorithm measure, we introduced two additional independent variables:

- The size of JPEG-100 compressed file, measured in MB: L_{JPEG};
- The corresponding compression ratio, calculated as the area S (in pixels) of the screenshot divided by the JPEG file size (in bytes): C_{JPEG}.

Procedure. The survey to collect data was implemented using LimeSurvey, and the participants used a web browser to interact with it. Some of them worked in university computer rooms, while the others used their own computer equipment with varying screen resolutions, to better represent the real context of use. Each subject was asked to evaluate the screenshots of the 21 websites' homepages (presented one by one in random order) per the 7 subjective scales. On average, it took each participant 30.3 minutes to complete the survey, and the data collection session lasted 19 days overall. We used screenshots, not the actual websites, to ensure uniformity of the experimental material between the participants.

4.2 Experiment Description (Layout Annotation)

Participants. In the second stage of the experiment, we employed 8 annotators (4 male, 4 female), whose ages ranged from 18 to 21 (mean 19.0, SD = 1.0). They were students of Novosibirsk State Technical University who volunteered to participate in this study as part of their practical training course. All the participants worked simultaneously in a same room, under instructor's supervision.

Design. The independent variables resulted from the annotation of the UIs:

- Number of rows in the overlaid grid: W_S_R;
- Number of columns in the overlaid grid: W_S_C;
- "Configuration" – the placement of 0s and 1s in the grid cells.

From these, we got the "derived" independent variables:

- Number of cells containing 1s: W_N;
- Number of all cells in the grid (W_S_R multiplied by W_S_C): W_S_0;
- Lengths of the row- and column-based bit strings compressed with the RLE-based algorithm: $W\_L_{RLE\text{-}R}$ and $W\_L_{RLE\text{-}C}$;
- Lengths of the row- and column-based bit strings compressed with the Deflate algorithm: $W\_L_{D\text{-}R}$ and $W\_L_{D\text{-}C}$;
- The corresponding data compression ratios for the algorithms: $W\_C_{RLE\text{-}R}$, $W\_C_{RLE\text{-}C}$, $W\_C_{D\text{-}R}$, $W\_C_{D\text{-}C}$.

Procedure. The screenshots of the websites were printed out in color on A4 format paper sheets (since the two websites were not found to be valid, the total number of annotated websites was 19). The participants were instructed to put the provided tracing paper A4 format sheets over the printed out screenshots and use the pencils to draw grid layouts marking 0 or 1 in each cell. The numbers of rows and columns in the grid were to be chosen individually by each annotator and each screenshot, depending on their impression of the layout design grid. However, the participants were told that more rows and columns are generally preferred over less. In the grid cells, 0s and 1s were also to be assigned subjectively, depending if the cell's interior was mostly whitespace (0) or interface elements/content (1). The order in which each of the 8 annotators processed the 19 screenshots was randomized. Some examples of the annotated websites are presented in Appendix B.

4.3 Descriptive Statistics

In the first stage of the experiment, we collected 1323 complexity evaluations for the 21 websites. Websites #9 and #14 were removed from further study due to technical problems with the screenshots (90.5% valid). In the second stage, we collected 152 annotations for the remaining 19 website screenshots. On overall, it took the annotators about 1 hour to finish their job.

The averaged value for SComplex (the complexity scale in this experiment ranged from 1 to 7) was 3.61 (SD = 0.77). The Pearson correlation between W_S_0 and W_N was highly significant ($r = 0.929$, $p < 0.001$).

We used t-tests to check statistical significance of differences in compression for row- and column-based strings. We found no statistically significant difference between $W\_L_{RLE\text{-}R}$ and $W\_L_{RLE\text{-}C}$ ($t_{151} = 1.222$, $p = 0.224$, $r = 0.807$). However, the difference between $W\_L_{D\text{-}R}$ and $W\_L_{D\text{-}C}$ was significant ($t_{4701} = -3.873$, $p < 0.001$, $r = 0.951$), the

mean lengths being 13.37 and 13.97 respectively. Still, we decided to use the "best" values for both algorithms (W_L_{RLE} and W_L_D), in correspondence with the experiment 1.

Again, we found statistically significant Pearson's correlations for W_L_{RLE} with W_L_D ($r = 0.941$, $p < 0.001$), $\log_2 W\_N$ with W_L_{RLE} ($r = 0.735$, $p < 0.001$), and $\log_2 W\_N$ with W_L_D ($r = 0.657$, $p < 0.001$). L_{JPEG} had no significant correlations (at $\alpha = 0.05$) with either of these three variables, but its positive correlation with W_S_0 was found to be significant ($r = 0.486$, $p = 0.035$), which suggests validity of this "screenshot length" measure provided by the annotators.

In this experiment, t-test showed no statistically significant difference between W_C_{RLE} and W_C_D ($p = 0.983$). However, C_{JPEG} was found to be significantly different (at $\alpha = 0.08$) from both W_C_{RLE} ($t_{18} = -1.86$, $p = 0.08$) and W_C_D ($t_{18} = -2.1$, $p = 0.05$).

4.4 Effects of Independent Variables

In Table 2 we show Pearson correlations between SComplex and the respective independent variables, but in this experiment none of the correlations were significant at $\alpha = 0.05$, which is explained in particular by the small sample size.

Table 2. Descriptive statistics and the correlations for the independent variables (experiment 2).

Variable	Range	Mean (SD)	r (SComplex)
W_S_0	22.25-54.75	34.22 (9.53)	0.245
W_N	12.5-32.00	21.82 (6.04)	0.343
$\log_2 W\_N$	3.64-5.0	4.39 (0.42)	**0.349**
W_L_{RLE}	6.75-22.5	15.14 (4.06)	0.149
W_L_D	8.25-17.38	12.94 (2.23)	0.269
L_{JPEG}	0.56-9.11	1.68 (1.95)	0.332
W_C_{RLE}	1.80-3.74	2.58 (0.57)	0.058
W_C_D	1.82-3.56	2.58 (0.48)	0.126
C_{JPEG}	1.19-3.18	2.22 (0.57)	-0.262

Just like in the first experiment, the strongest correlation with complexity ($r = 0.349$) was found for $\log_2 W\_N$. The correlation for L_{JPEG} that we considered as the baseline was somehow weaker ($r = 0.332$).

Hypothesis #2 couldn't be validated by strict analogy with the first experiment, since in the second one there was no significant difference between the compression ratios provided by RLE and Deflate algorithms. But we should note that while JPEG algorithm compression ratio was significantly lower on average, C_{JPEG} had stronger correlation ($r = -0.262$) with SComplex, in comparison with W_C_{RLE} and W_C_D.

Additional analysis was performed to explore relations between the number of elements specified by participants of the subjective complexity evaluation stage and the annotators' results. We found significant correlations between SElements and W_N

($r = 0.719$, $p = 0.001$), which was stronger than the one between SElements and W_S_0 ($r = 0.562$, $p = 0.012$). The correlation between SWhite and the calculated whitespace measure ($1 - W\_N/W\_S_0$) was also significant ($r = 0.531$, $p = 0.019$). These results suggest that whitespace was mostly annotated as grid cells with 0s, while the interface elements visible to the complexity evaluators were annotated as grid cells with 1s.

4.5 Regression Analysis

Again, first we performed regression analysis with the baseline factors. The model with $\log_2 W\_N$ was not significant ($p = 0.143$) and had $R^2 = 0.122$. The model with L_{JPEG} had lower significance ($p = 0.164$) and $R^2 = 0.110$. Notably, the regression models for the other factors we considered (W_S_0, W_N, W_L_{RLE}, W_L_D) had even lower significances and R^2 values.

Further, we attempted regression with all 9 independent variables (see in Table 2) as factors. We used Backwards variable selection method, which now led to the model with three significant factors (at $\alpha = 0.07$): $\log_2 W\_N$ (Beta = 0.619, $p = 0.067$), W_L_{RLE} (Beta = -1.534, $p = 0.043$), W_L_D (Beta = 1.306, $p = 0.054$). This model had $R^2 = 0.339$ ($F_{3,15} = 2.57$, $R^2_{adj} = 0.207$, AIC = 43.2):

$$SComplex = -2.85 + 1.15 \log_2 W\_N - 0.29 W\_L_{RLE} + 0.45 W\_L_D \quad (9)$$

However, among the intermediate models considered within the Backwards selection, there was a model with higher $R^2_{adj} = 0.263$ and lower AIC = 42.5. The factors in this model ($F_{4,14} = 2.61$, $R^2 = 0.427$) were $\log_2 N$ (Beta = 0.498, $p = 0.134$), W_L_{RLE} (Beta = -1.694, $p = 0.026$), W_L_D (Beta = 1.471, $p = 0.031$), and L_{JPEG} (Beta = 0.328, $p = 0.165$):

$$\begin{aligned} SComplex = {} & -2.37 + 0.92 \log_2 W\_N - 0.32 W\_L_{RLE} + \\ & + 0.51 W\_L_D + 0.13 L_{JPEG} \end{aligned} \quad (10)$$

5 Conclusion

In our paper we examined perceived visual complexity of UI layouts, which are a major controlled feature for every human-computer interface designer. Particularly, we sought to introduce data compression algorithms, which with respect to complexity have both solid theoretical justification (AIT) and wide practical application (JPEG-based metrics). So, we proposed the UI layouts coding method and employed the RLE and Deflate algorithms to produce compressed strings. The chosen algorithms are well established in image compression and presumably have advantage over information entropy-based compressibility measures. Our analysis of the experimental data suggests the following conclusions per the hypotheses formulated in the study:

Hypothesis #1. The compressed strings' lengths do explain layouts' complexity perception in humans, as the corresponding factors were significant in regressions for

the model (7) and real UIs (9). However, the correlations (Table 1 and 2) suggest that the Hick's law factor is even stronger connected to the perceived layout complexity.

Hypothesis #2. Unexpectedly, algorithms that showed higher compression ratios had weaker connection to the perceived complexity (Table 1 and 2). L_D was not significant in regression (7), unlike L_{RLE}, while L_{JPEG} had the lowest significance in (10).

Hypothesis #3. Generally, matrix conversion to string representation was no different by rows or by columns, with the only exception of the 4.4% difference for Deflate algorithm in experiment 2, where the sample size was relatively small.

Hence, we see the main contributions of our paper as the following:

1. We proposed the "squint" coarsening method for coding UI layouts into binary strings and demonstrated its use with real web interfaces. Practical applicability of the method is supported by highly significant correlation ($r = 0.719$) between the subjectively assessed number of interface elements and the number of annotated cells containing 1s, as well as significant correlation ($r = 0.531$) between the subjective amount of whitespace and the share of annotated cells containing 0s.
2. We demonstrated that compression algorithms can provide metrics that improve prediction of perceived VC of UI layouts. At the same time, we found that the classical Hick's law informational metric is well applicable as the main factor, which may imply its certain "revival" in HCI.
3. We found that better compression algorithms, which presumably come closer to the "real" Kolmogorov complexity, do not explain layouts VC perception better. It may mean that layouts are a specific type of visual objects [24].
4. We found that the way two-dimensional matrix representing the modelled UI layout is converted into bit string (by rows or by columns) does not affect the compression measure's explaining power with regard to layouts VC.

We see the main limitation of our study in employment of ordinal scales to measure VC of both model representation and web UIs. We plan to develop the approach for using interval scale, presumably based on tasks performance time. Also, the sample size in the websites experiment was relatively small, so the found statistical effects were not always convincing. Our further prospects include software implementation of the proposed coarsening method, so that coded representations for web UI screenshots could be collected automatically. We also plan to explore whether the coarsening method that we proposed for WUI layouts, could be suitable for producing coded representations for other types of visual objects.

Acknowledgement. The reported study was funded by Russian Ministry of Education and Science, according to the research project No. 2.2327.2017/4.6.

References

1. B. Castellani. Brian Castellani on the Complexity Sciences. Theory, Culture & Society, Oct. 2014 (2014).

2. Reinecke, K. et al.: Predicting users' first impressions of website aesthetics with a quantification of perceived visual complexity and colorfulness. Proc. of the ACM SIGCHI conference on human factors in computing systems, 2049-2058 (2013).
3. Machado, P. et al.: Computerized measures of visual complexity. Acta psychologica, 160, 43-57 (2015).
4. Michailidou, E., Harper, S., Bechhofer, S.: Visual complexity and aesthetic perception of web pages. In Proc. of the 26th ACM Int Conf on Design of communication, 215-224 (2008).
5. Taba, S.E.S. et al: An exploratory study on the relation between user interface complexity and the perceived quality. In Int. Conf. on Web Engineering, 370-379. Springer (2014).
6. Wu, O., Hu, W., Shi, L.: Measuring the visual complexities of web pages. ACM Transactions on the Web (TWEB), 7(1), p.1 (2013).
7. Chikhman, V. et al: Complexity of images: Experimental and computational estimates compared. Perception, 41(6), 631-647 (2012).
8. Alemerien, K., Magel, K.: GUIEvaluator: A Metric-tool for Evaluating the Complexity of Graphical User Interfaces. In SEKE, 13-18 (2014).
9. Stickel, C., Ebner, M., & Holzinger, A.: The XAOS metric–understanding visual complexity as measure of usability. In Symposium of the Austrian HCI and Usability Engineering Group, 278-290. Springer, Berlin, Heidelberg (2010).
10. Miniukovich, A., De Angeli, A.: Quantification of interface visual complexity. In Proc. of the 2014 ACM Int. Working Conf. on advanced visual interfaces, 153-160 (2014).
11. Donderi, D. C.: Visual complexity: a review. Psychological bulletin, 132(1), 73 (2006).
12. Yu, H., Winkler, S.: Image complexity and spatial information. In IEEE Fifth Int. Workshop on Quality of Multimedia Experience (QoMEX), 12-17 (2013).
13. Solomonoff, R.: The application of algorithmic probability to problems in artificial intelligence. In Machine Intelligence and Pattern Recognition, 4, 473-491 (1986).
14. Rosenholtz, R., Li, Y., Nakano, L.: Measuring visual clutter. J. of Vision, 7(2), 1-22 (2007).
15. Carballal, A. et al.: Distinguishing paintings from photographs by complexity estimates. Neural Computing and Applications, 1-13 (2016),
16. Chang, L.Y., Chen, Y.C., Perfetti, C.A. GraphCom: A multidimensional measure of graphic complexity applied to 131 written languages. Behavior res. methods, 50(1), 427-449 (2018).
17. Heil, S., Bakaev, M., Gaedke, M.: Measuring and ensuring similarity of user interfaces: the impact of web layout. Lecture Notes in Computer Science, vol. 10041, 252-260 (2016).
18. Comber, T., Maltby, J.R.: Layout complexity: does it measure usability?. In Human-Computer Interaction INTERACT'97, 623-626. Springer, Boston, MA (1997).
19. Michalski, R., Grobelny, J., & Karwowski, W.: The effects of graphical interface design characteristics on human–computer interaction task efficiency. International Journal of Industrial Ergonomics, 36(11), 959-977 (2006).
20. Seow, S.C.: Information theoretic models of HCI: a comparison of the Hick-Hyman law and Fitts' law. Human-Computer Interaction, 20(3), 315-352 (2005).
21. Kim N.W. et al. BubbleView: an interface for crowdsourcing image importance maps and tracking visual attention. ACM Transactions on Computer-Human Interaction (TOCHI), 24(5), Art. 36 (2017).
22. Xu, P., Sugano, Y., Bulling, A.: Spatio-temporal modeling and prediction of visual attention in graphical user interfaces. Proc. of the ACM CHI Conference on Human Factors in Computing Systems, 3299-3310 (2016).
23. M. Bakaev et al.: HCI Vision for Automated Analysis and Mining of Web User Interfaces. Proc. Int. Conf. on Web Engineering (ICWE), 136-144. Springer, Cham (2018).
24. Simon H.A.: Complexity and the representation of patterned sequences of symbols. Psycho-logical review, 79(5), 369 (1972).

Appendix A

The *encode* function implements simple RLE algorithm compression. The input variable for the function is binary string.

```
function encode($input)
{
  if (!$input) {
    return '';
  }

  $output = '';
  $prev = $letter = null;
  $count = 1;
  foreach (str_split($input) as $letter) {
       if ($letter === $prev) {
         $count++;
       } else {
         if ($count > 1) {
           $output .= $count;
         }
         $output .= $prev;
         $count = 1;
       }
       $prev = $letter;
  }

  if ($count > 1) {
    $output .= $count;
  }

  $output .= $letter;
  return $output;
}
```

Appendix B

In Fig. B.1 we demonstrate the annotation process, in which coded representations were provided for the web UIs. Fig. B.2 shows screenshot of website #2 (a German university), while in Fig. B.3 the same screenshot is presented with the annotated tracing paper. Fig. B.4 and Fig. B.5 provide the same illustrative example for website #21 (a Russian university).

Fig. B.1. The annotators working on the printed out websites screenshots.

Fig. B.2. Initial screenshot of website #2 homepage.

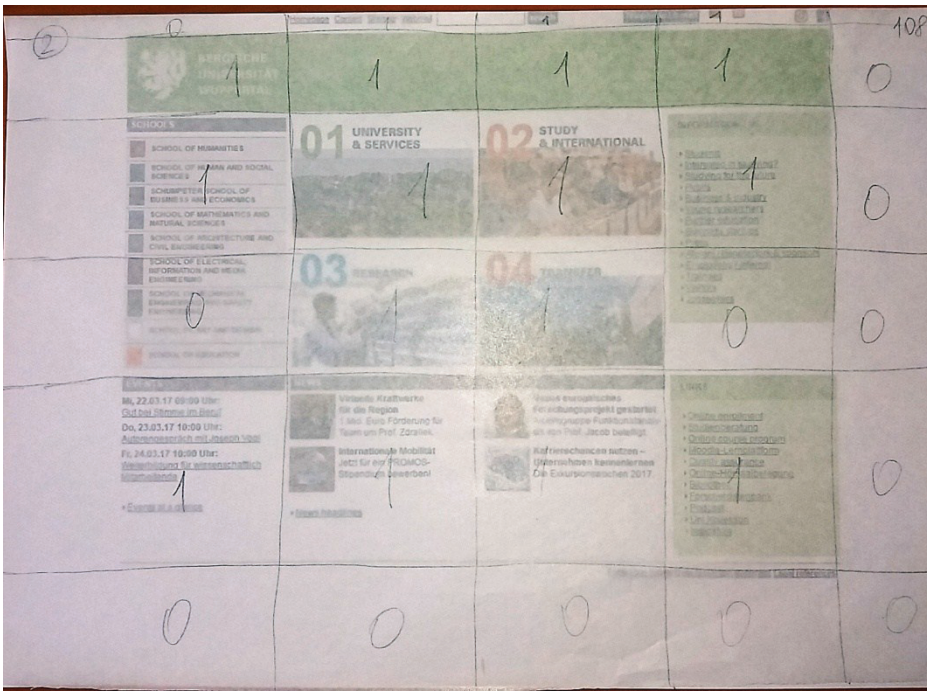


Fig. B.3. An annotated screenshot for website #2.

Fig. B.4. Initial screenshot of website #21 homepage.

Fig. B.5. An annotated screenshot for website #21.

Case-Based Genetic Optimization of Web User Interfaces

Maxim Bakaev [0000-0002-1889-0692] and Vladimir Khvorostov

Novosibirsk State Technical University, Novosibirsk, Russia
bakaev;xvorostov @corp.nstu.ru

Abstract. The combination of case-based approach and genetic optimization can provide significant boost to effectiveness of computer-aided design of web user interfaces. However, their integration in web design domain requires certain sophistication, since parts of available solutions cannot be reused directly, due to technical and legal obstacles. This article describes evolutionary algorithm for automatic generation of website designs, which treats parameters of functionality, layout and visual appearance as the variables. The structure of the chromosome is devised, allowing representation of websites' properties in the above three manipulated aspects and facilitating easy application of the genetic operators. The authors also perform feature engineering for web projects and construct the case base as well as supplementary repositories of functional components and filler-up content. The implementation of the approach is demonstrated with the means provided by the popular Drupal web framework. The results of the study can empower case-based reuse of existing web designs and therefore be of interest to both AI researchers and software engineers.

Keywords: Web User Interface Design, Case-Based Reasoning, Software Components, Drupal Framework.

1 Introduction

The continuing exponential growth in the amount of data is accompanied by increase in diversity of data sources and data models. This, together with the forthcoming "Big Interaction", with its multiplicity of user tasks and characteristics, of interface devices and contexts of use, may soon render hand-making of all the necessary human-computer interfaces unfeasible. Discrete optimization methods are seen quite promising for intelligent computer-aided design of interaction, but the combinatorial number of possible solutions is huge even for relatively simple user interface (UI) design problems (Oulasvirta, 2017). Increasingly, genetic algorithms are used as effective tool for solving optimization problems, and there are reports of their successful use for conventional websites designs (Qu, 2015), whose UIs are not particularly creative, but mostly provide data I/O.

Genetic algorithms are based on repeated application of the genetic operators – generally these are selection of candidates for reproduction, crossover (producing child solutions incorporating features of numerous parents) and mutation (introduction of random or directed variations in the features). Their superset is evolutionary

algorithms (EAs), which are being successfully used in programming, engineering, website design (Guo et al., 2016), data mining and classification (Freitas, 2013), natural language processing, machine learning, intelligent and recommender systems, as well as many other domains (Coello, 2015). One of the most popular methods for website design improvement, A/B Testing, which measures performance of slightly different design versions in the real environment, is essentially an informal EA as well. A/B Testing does not generally employ a genetic representation of designs, and one of its weaknesses is lack of a systematic approach to recombination, so that the method always risks turning into a random walk.

Most types of EAs rely on special data structures that represent properties of a candidate solution – chromosomes, in which concrete values comprise the solution's genotype. Individual genes that constitute the chromosome may be of different types, depending of a particular problem, and the genetic operators need to take into consideration the types' boundaries and the allowed values – alleles. Hence, just as design of data structures is of crucial importance in software engineering, the choice of the chromosome structure can remarkably affect EA's convergence, speed and the end result's quality (Michalewicz & Hartley, 1996).

So, an archetypal EA incorporates the following stages:

1. Initialization – creation of the first generation (initial population), which can be either random or based on some distinguished existing cases (Kazimipour et al., 2015), for better convergence to global optimum or faster performance.
2. Selection of best-fit individuals for reproduction (parents) based on evaluation of their fitness, which is performed either in real world context or, more commonly, with specially formulated fitness functions.
3. Reproduction of parents to create individuals of the next generation, in which mutation and crossover are used. If the algorithm's termination condition (sufficient quality of the final solution or time limit) is not met, it goes back to the previous step and repeats.

Fitness function (FF) is central for the ultimate EA success: it must both fully represent the optimization goal and be as easily computable as possible. In domains that deal with human preferences and capacities, such as WUI design, specification of FF is far from trivial, especially given the diversity of user characteristics and tasks. Interactive Evolutionary Computation, which basically delegates FF evaluation to humans, may slow down the algorithm considerably, so pre-trained user behavior models are often seen as a solution (Bakaev et al., 2017b). Yet another way to represent the ever-changing "environment" that the candidate designs must fit to, is exploiting operating websites with that the users actually interact. The assumption here is that surviving web projects have more or less successfully adapted to their target users' preferences and needs. The degree of the successfulness can be automatically estimated from website's interaction statistics and popularity, representing the dynamic quality-in-use (Bakaev et al., 2017a), while the static quality-as-is can be assessed based on static design metrics (Ivory & Hearst, 2002). However, directly employing these well-fit examples as parents in the EA and reusing their parts is rather problematic in the web design domain, due to technical (lack of access to the website's back-

office and server-side code) and legal (copyright protection) reasons. Instead, measure of genotypic or phenotypic similarity with the example solutions can be included as one of the components in the FF. A notable example of such approach was once implemented in SUPPLE intelligent system for UI automatic generation (Gajos et al., 2005): the optimized goal function could incorporate the metric of similarity with the previous version of UI design – in SUPPLE's case, to maintain familiarity for its users. The metric was linear combination of pairwise similarities between interface widgets, whose features included language, orientation, geometry, etc.

The number of operational websites accessible on the World Wide Web is currently estimated as 100-250 millions, so there should be no shortage of well-fit design examples for any kind of target users and tasks. The problem is actually the opposite: even despite the recent emergence of design mining field that focuses on extraction of design patterns and trends from large collections of design examples, there is lack of repositories or services capable of finding existing solutions relevant to a new project's UI design context. We believe that the problem could be resolved by supplementing the WUI evolutionary optimization algorithm with case-based reasoning (CBR) approach that has shown successful use in both software and web engineering (Rocha et al., 2014; Renzis et al., 2016), WUI development (Marir, 2012), as well as in many non-IT domains. More detailed justification on CBR applicability for WUI design can be found in one of our previous works (Bakaev, 2018a). The classically identified stages in CBR can be summarized as follows (Mantaras et al., 2005):

- Retrieve: describe a new problem and find similar problems with known solutions in a case base (CB);
- Reuse: adapt the solutions of the retrieved problems to the current problem, in an assumption that similar problems have similar solutions;
- Revise: evaluate the new solution and possibly repeat the previous stages;
- Retain: store the results as a new case.

Case thus equals parameterized problem plus one or several solutions, each of which can be supplemented with quality – i.e. how good the solution was in resolving the problem. The retrieval stage is arguably what most today's CBR-related research focuses on, and to the extent of our knowledge there is lack of techniques for adaptation of the retrieved WUI designs. Or rather, due to the technical and legal impediments we've mentioned above, it's the new solution that has to be adapted to the exemplary designs. For instance, the database of the compelling *design mining* Webzeigeist tool (Kumar et al., 2013) that implemented a kind of design scraping and searching engine, presumably contained millions of web pages. However, it allows searching by technicalities, such as page aspect ratio or element styles, but not by domain- or user-related aspects. This, effectively, does not allow supporting the CBR approach, is no designs appropriate for a particular problem specification can be retrieved. The same holds for web design frameworks; and while they provide rich libraries of interface elements (icons, buttons, form fields, etc.), neither they can aid in adaptation of a chosen design for a particular user group or to resemble exemplary solutions.

Our paper is dedicated to integration of case-based reasoning and evolutionary approaches in web design, which we consider necessary for practical feasibility of UI

optimization in this design field. Particularly, we focus on development of the case base and the data structures that the EA relies on. In Section 2, we outline the proposed EA for CBR-based web design and justify our approach to genetic representation of the solutions. In Section 3, we highlight some particulars of the implementation, specify the concrete chromosome structure, and provide some examples. In the Conclusion, we summarize the contribution of our article and provide final remarks.

2 Methods

2.1 The Evolutionary Algorithm

The combination of the case-based reasoning approach and the genetic optimization algorithm for WUI design (for detailed description and justification of the algorithm see Bakaev & Gaedke, 2016) can be outlined in the following process:

1. Designer, who initiates the process, creates a new web project (case) in the CB and specifies the problem features and other input information. We will address this in the subsequent sub-section.
2. The CBR algorithm retrieves relevant projects from the CB: based on similarity measures for the cases and the assessed quality of the currently operational solution (website version). We outline the retrieval process based on problem similarity in Section 3.
3. The EA (re-)produces new solutions (web designs), dealing with the following major aspects of a web design:

   - Functionality: since the EA is suitable for creating relatively simple websites, we should auto-generate not the actual programming code, but rather assemble the available pieces of web functionality, saving their configuration in the chromosome. We justified and detailed the use of functional components through their meta-repository in (Bakaev, 2018c). In Section 3 of the current paper we demonstrate their specification in the chromosome.
   - Content: by definition, web content is very changing and it does not actually relate to a website's structure, although it may be perceived as related to its design by users. So, it makes little sense to store the content-related properties in the chromosome, they should rather correspond to the phenotype – the individual solutions' properties, which can vary for the same genotype. The content repository is described in Section 3.
   - Page structure (layout): in modern websites, pages are composed from elements that are organized hierarchically and consistently ordered within their siblings. This is essential part of the design and its usability, and the corresponding information needs to be saved in the chromosome, which we also describe in the subsequent sub-section.
   - Visual appearance: it is potentially the vastest part of the design space, since the combinatorial number of all possible colors, font styles and sizes, etc. is enough to make every website out of the existing billion unique. The elements do have

constraining relations between them (e.g. font's and background's colors must provide enough contrast), but attempts to specify formal rules for web design have so far been rather fruitless in practice. So, the initial values are quite important for the EA's convergence, and the algorithm should better start from a reasonable visual solution. We representation of the visual appearance in the chromosome is demonstrated in Section 3.

4. The EA evaluates fitness functions for the new solutions, based on their similarity with the reference solutions in the retrieved projects and quality assessed with the pre-trained target users' behavior models, and selects the best fit solutions. The corresponding approaches were detailed in (Bakaev, 2018a) and (Bakaev, 2017b).
5. If the EA's finishing conditions aren't met, the algorithm applies the genetic operators and goes again to step 3 to create new generation of solutions, with the new genotypes. We provide an example of applying the mutation genetic operator to website's visual appearance in Section 3.
6. The CBR algorithm retains the best solution(s) produced by the EA in the CB, specifying it as prototype. If the web project later goes live, the CB daemons start collecting the quality attributes for the solution (website), e.g. on the basis of the Web Intelligence approach that we previously proposed (Bakaev, 2017a).

So, in the subsequent sub-section we consider the specificity of the problems and solutions for CBR in the WUI design domain.

2.2 CBR: the Problem Features and the Solutions' Chromosome Structure

Devising the accurate structure to represent the problem's properties is seen as crucial for machine learning and automated reasoning tasks (Anderson et al., 2013). With respect to CBR, this feature engineering also plays important part in shaping the structure of the CB. The process generally includes: forming the excessive list of potential features, implementing all or some of them in a prototype, and selecting relevant features by optimizing the considered subset. In our feature engineering for WUI design problem, we considered web project as corresponding to a case (as design- and goal-wise complete entity) and website to a solution (as several versions of operational and prototype websites are often created in attempts to meet the web project's goals). We relied upon three models selected from the ones prescribed in WUI development: Domain, Tasks, and User (the Platform and Environment models were excluded since they rather relate to website's back-office). The detailed description of the feature engineering for web projects can be found in (Bakaev, 2018a).

Unlike reusable programming code, existing website designs differ dramatically in eminence, so the quality aspects must be stored for the solutions, to be considered in the retrieval in addition to similarity. Website quality is best described as collection of attributes, whose relative importance can vary depending of the particular project's goals and context (Glass, 2002). Thus, the set of quality-related features must be extendable and provide flexibility for different formulations of the overall quality function. The quality-related values for the cases collected from the WWW (i.e. for someone else's websites) can be obtained e.g. based on the Web Intelligence approach (Bakaev, 2017a).

As we mentioned before, the key data structure in EA is chromosome, which contains code for the important properties of solutions. Most traditionally, EAs just use linear binary representations for chromosomes, particularly in web design (Qu, 2015). However, this implies that knowledge about the design space (the interrelations between the genes) has to be delegated to the procedures responsible for reproduction, crossover, etc.; otherwise the EA may end up trying combinatorial matches and lose in the convergence speed. So it is sensible to separate out this knowledge in an appropriate data structure – domain ontology effectively representing the design space for WUI. Properties (attributes) of the ontology classes thus correspond to genes, while their datatypes and alleles, crucial for the genetic operators' proper application, are defined via the facets' values. WUI design support ontology that we developed in the popular Protégé-Frames editor can be used to represent the functionality, layout and visual appearance. In Fig. 1, we show a fragment of the ontology with some classes related to the genes responsible for web page's visual appearance, relying on the accepted CSS specification.

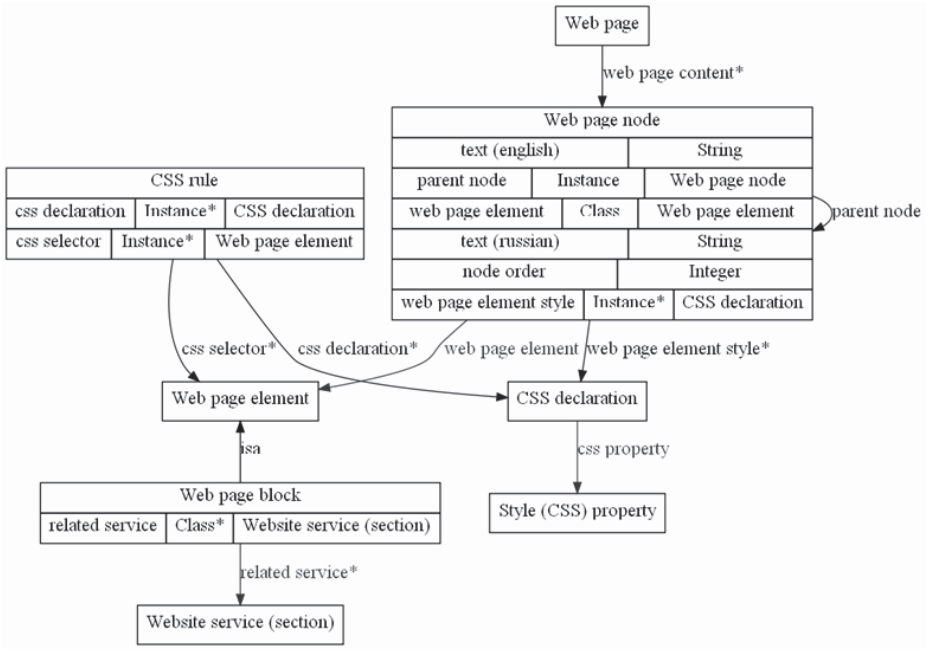


Fig. 1. Ontology classes and properties for a webpage layout and visual appearance.

3 Implementation

In this section we present some highlights of the proposed approach that we implemented (see at our dedicated portal (http://wuikb.online). We used Drupal web content management framework as the platform for the implementation. Despite the usual drawbacks associated with the use of frameworks, such as lower performance and flexibility, the following advantages motivated our choice:

- Drupal has robust architecture that allows handling high number of components.
- Drupal has lots of components (modules) ready for reuse, they are well-organized and centralized in the single repository with API access (http://drupal.org), they have auto-maintained quality attributes (# of downloads, actual installs, open bugs, etc.).
- Drupal has programmable (via command line, API, etc.) support for installment of websites, the layout of interface elements on webpage, handling web forms, menus, content items, adjustment of visual appearance styles, and so on.

3.1 The Case-Based Retrieval

The case base is the registry of projects each of which correspond to a case and can be either automatically scrapped from the web or specially created (Bakaev, 2018c). The problem description are Domain, User and Task features, whose values are either directly specified when a new project is created, mined by the supplementary tools, or provided by human annotators. As conventional websites of the same domain have fairly predictable functionality, there was no need to employ full-scale task modeling notations, such as W3C's CTT or HAMSTERS (Martinie et al., 2015). So, the Task model is represented as the structured inventory of website chapters, for which similarity can be calculated fairly easily (see example in Fig. 2). The solutions (websites) have quality attributes, which are either provided by users or experts, or obtained automatically by the supplementary tools in the course of the CB population. So, the CBR algorithm retrieves the cases from the CB based on the following sequence:

1. Pre-selects a set of cases based on the domain similarity
2. For each case in the set it calculates the distance measure that incorporates domain similarity, task similarity and target user similarity (see in Bakaev, 2018a).
3. For the most similar cases it calculates the normalized quality values in the range (0;1) and retrieves similar cases with the highest qualities.

After the retrieval of cases, the "classical" CBR prescribes adapting their solutions, but as we noted above, in web design this process (the Reuse and Revise stages) can't be performed directly due to legal and technical obstacles. Instead, the EA will consider similarity with the retrieved solutions as part of the new solutions' fitness.

3.2 The Chromosome Structure Specification

Our design of the chromosome structure for the EA was based upon the following previously justified theoretical premises and technical considerations:

1. there are three distinct dimensions of a website: functionality, layout (page structure) and visual appearance;
2. representation of design space-related knowledge should be minimized in the code implementing the genetic operators, but moved to the chromosome instead;
3. the chromosome structure design should allow maximal use of the means provided by the framework (without relying on its GUI), particularly Drush project of Drupal.

Based on the above, instead of the classic binary strings we have chosen to rely on the popular "name: value" representation for each of the genes, since it also allows ex-

pressing classes and properties from the developed ontology. We subsequently use the Backus-Naur form to describe the parts of the chromosome per the three website dimensions. In one of the following sub-sections we also provide example of using chromosome to specify website features.

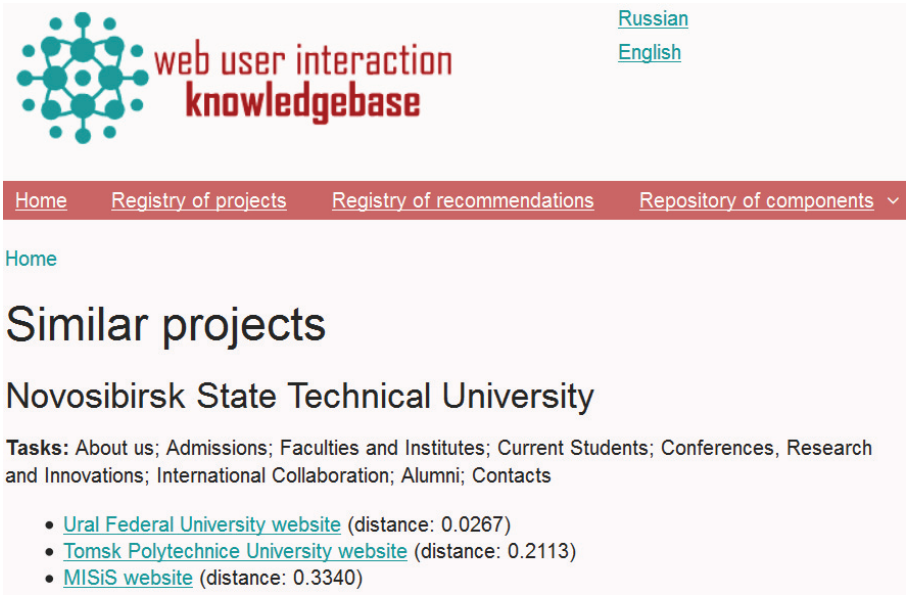


Fig. 2. Retrieved cases based on Task similarity (screenshot from http://wuikb.online)

The representation of website functionality in the chromosome is based on Drush's makefile syntax (http://docs.drush.org/en/8.x/make/) that in turn corresponds to YAML. The website Domain is selected from DMOZ classification. Task names come from the Tasks model, and Drupal modules (themes are not allowed, unlike in the makefile) implement the tasks on a many-to-many relationship basis. Custom configuration for a component can be stored in the string that concatenates the project options from the makefile (by default, most recent production versions of the modules are used). Since the functionality install is the most time-consuming action in the whole EA, the genetic operators are only applied to the modules if the alternatives (alleles) have comparable quality. The resulting data structure in the chromosome can be specified as follows:

```
<functionality_genes> := <domain> | <functionality_item>
| <component_configuration>
<domain> := "domain" ": " <dmoz_domain_name>
<functionality_item> ::= <task_name> ": " <dru-
pal_module_name> | <functionality_item>
<component_configuration> ::= "_" <drupal_module_name> ":
" <configuration_string> | <component_configuration>
```

Generally, each functionality item implemented for the website's front office has one or several related user interface elements. Correspondingly, in Drupal most front-office modules have blocks (groups of UI elements) placed in one of webpage's regions and ordered within a region by their weight. The currently default list of regions in Drupal (Twig template engine) has 10 items: page.header, page.primary_menu, page.content, etc., which are sufficient for many conventional websites. The names of the blocks are created by Drupal, so they become accessible for the EA after the website functionality is assembled. The part of the chromosome describing the layout can then be specified as follows:

```
<layout_genes> ::= <blocks_placement> | <blocks_order>
<blocks_placement> ::= <region_name> ": " <block_name> |
<blocks_placement>
<blocks_order_in_region> ::= <block_name> ": " <weight> |
<blocks_order_in_region>
<region_name> ::= "page.header" | "page.primary_menu" |
"page.secondary_menu" | "page.highlighted" | "page.help"
| "page.content" | "page.sidebar_first" |
"page.sidebar_second" | "page.footer" | "page.breadcrumb"
```

The genetic operators are applied to the blocks' placement and order, which can be programmatically set via Drush extras commands, such as:

```
drush block-configure --module=block --delta=block_delta
--region=page.header --weight=10
```

In Drupal, themes are responsible for visual appearance of the website, and many of them have adjustable parameters (stored in JSON format) shaping their visual presentation, such as colors, font sizes and families, etc. In the EA, after the website functionality and layout genes are initialized, the algorithm selects the basic theme that has the number of parameters appropriate for the project (more parameters mean more flexibility, but EA's convergence may take much longer time). The visual appearance-related part of the chromosome can be specified as follows:

```
<visual_appearance_configuration> ::= <theme_name>
<theme_parameters>
<theme_name> ::= "theme" ": " <drupal_theme_name>
<theme_parameters> ::= | <parameter_name> ": " <parame-
ter_value> | <theme_parameters>
```

The genetic operators can be applied to the parameter values, and for each new solution the corresponding sub-theme is created. Although the theme parameters are mechanism very native for Drupal, there's no built-in universal solution for managing the parameters. So, we implemented the dedicated Drush module to adjust themes' parameters from command line (see https://github.com/vkhvorostov/subtheme_color).

3.3 Repository of Content

In the eyes of end users, content is not entirely detachable from design, so auto-assessment of the candidate solutions' quality in the EA's fitness function cannot be performed with wireframe designs, lacking any textual and graphic materials. Obviously, the assessment of the solutions' similarity with the CBR-retrieved designs also calls for filler-up content resembling the one of the reference websites. As we mentioned before, there's barely the need to store content-related properties of solutions in the chromosome (they should rather relate to phenotype). Thus, content items are to be drawn from the respective repository and assigned to the solutions on a stochastic basis. The concrete use cases for such repository are the following:

1. Extracting "original" content items from solutions in projects:
   a. manually by human annotators;
   b. automatically from the solutions' webpage code and screenshot (our corresponding supplementary tool, the visual analyzer, is described in Bakaev et al., 2018b).
2. Creating filler-up content items (linked to the original ones):
   a. manually by human annotators;
   b. automatically using online services for similar images search, text generators, etc.
3. Manual organization and management of the content items.
4. Usage of the content by the EA:
   a. drawing the filler-up content items similar to the original ones in the retrieved projects;
   b. considering the degree of similarity in defining the probability for selecting a particular filler-up content item;
   c. considering the content type and placement in the webpage.

Correspondingly, the Content Item has the following attributes (see in Fig. 3):

- type (text, image, label, header, etc.);
- status (original, filler-up, outdated, etc.);
- content (piece of HTML code);
- links to Projects (mostly for filler-up content) and to their solutions (mostly for original content);
- links to other Content Items (with weights indicating of the degree of similarity).

In Fig. 4 we show a) part of a reference website's design, b) an extracted content item – logo; c) related filler-up content – a similar image collected via Google, with color similarity constraint (red); d) related filler-up content – a similar image collected via Google, without the color similarity constraint.

3.4 Example of the EA Data Structures

In the example we show some data structures from genetic optimization of an educational website project. Extract from the part of chromosome responsible for the functionality generation is presented below (its specification was previously described in

The Chromosome Structure sub-section). “Drupal” module name implies that functionality is in Drupal core.

```
domain: "Career and Education"
about_us: drupal
_drupal: "version 7.59"
contact_us: webform
shopping_cart: dc_cart_ajax
...
```

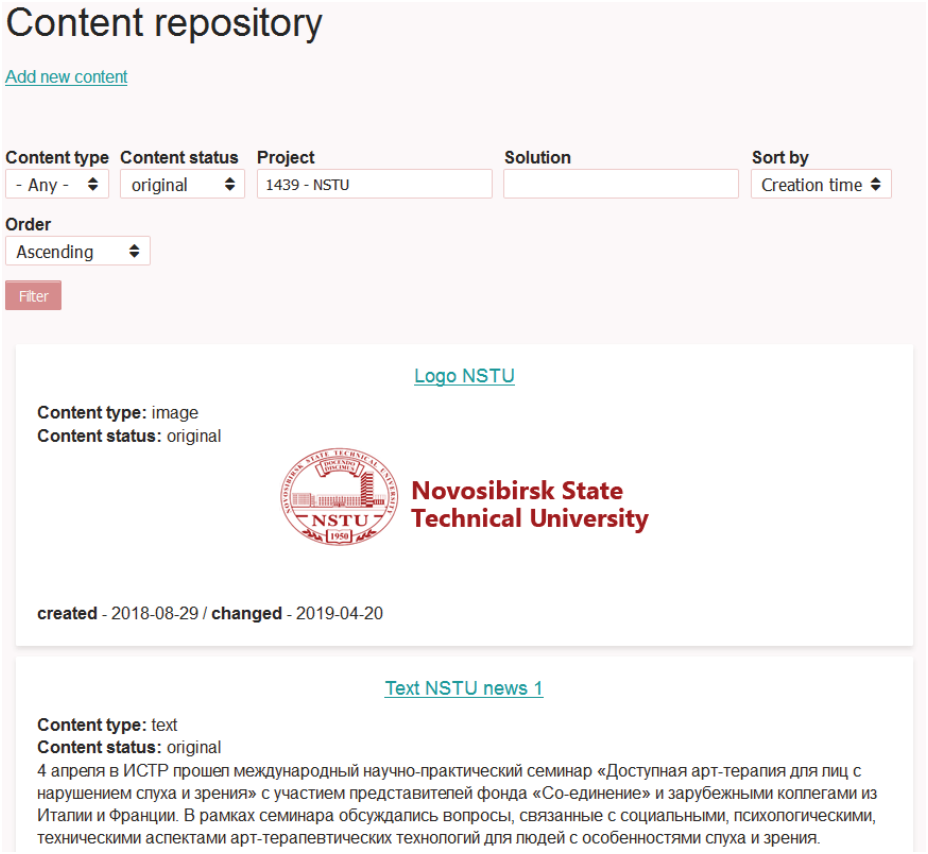


Fig. 3. The list of Content Items in the repository (screenshot from http://wuikb.online)

The Component picker also produces the list of alleles for the shopping_cart task (with the quality value being the number of websites using the module) outside of the chromosome:

```
shopping_cart:
  dc_cart_ajax: 1874
  commerce_ajax_cart: 1375
```

```
  basic_cart: 1288
  uc_ajax_cart: 1278
...
```

Subsequently, the Website installer additionally installs Drupal's commerce module (as dependency for dc_cart_ajax module).

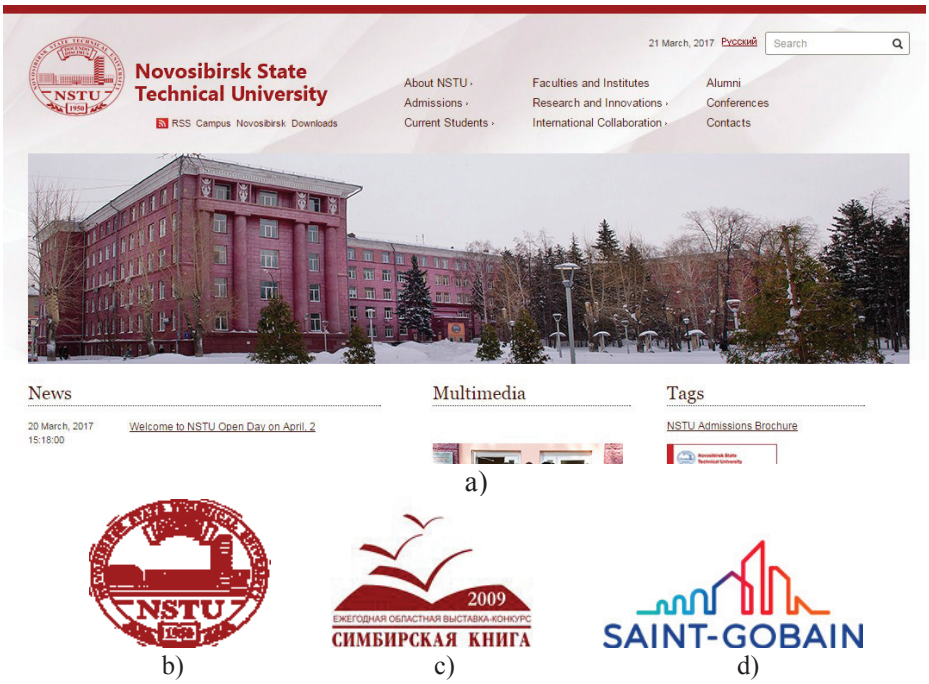


Fig. 4. The related items in the content repository

The layout part of the chromosome is naturally created after the functionality part, and we present the extract below:

```
page.header: site-logo
page.header: site-name
site-logo: -10
site-name: 10
page.primary_menu: topbar
page.sidebar_first: content-block-1
page.content: user-login
page.footer: content-block-2
...
```

The number of genes responsible for the visual appearance is potentially unlimited and defined by the theme parameters. An extract of the chromosome's corresponding part is shown below:

```
theme: bartik
top: '#dada4a'
bg: '#ffffff'
footer: '#161617'
text: '#3b3b3b'
link: '#0073b6'
...
```

Our subtheme_color module generated the following command to mutate two of the above genes:

```
/drush.sh -r projects/test/ stc bartik '{"top":
"#daba4a", "footer": "#361617"}'
```

In Fig. 5 we illustrate the mutation, showing the visual appearance before and after application of the genetic operator.

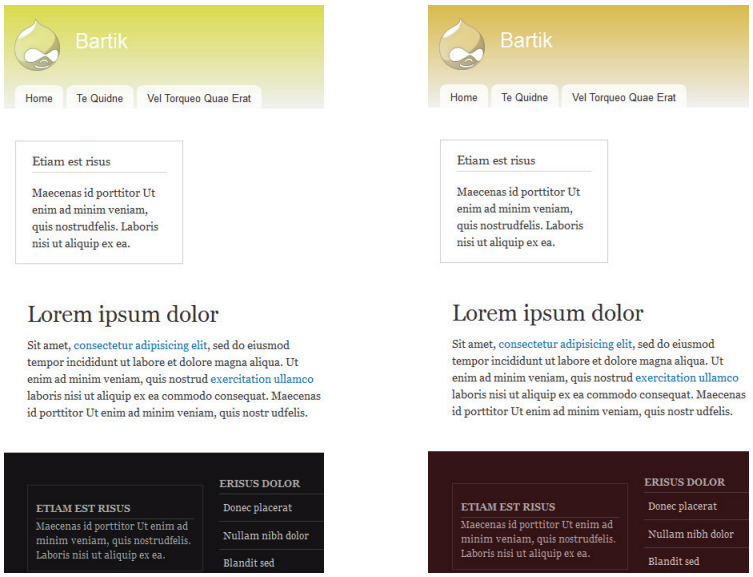


Fig. 5. Theme's mutated visual appearance in generation N (left) vs. N+1 (right).

4 Conclusion

Case-based reasoning can provide a significant boost to effectiveness of computer-aided optimization-based design of web UIs. Their integration requires certain sophis-

tication, since parts of available solutions cannot be reused directly, due to technical and legal obstacles. We addressed this and other particulars of the proposed approach and see the main contributions of the current work as follows:

1. We performed feature engineering for web projects, inspired by the popular model-based approach to web interface development. Further, we demonstrated construction of the CB, together with repositories supplementing functionality- and content-related dimensions.
2. We devised the stages of the EA in the considered field and designed the chromosome structure to represent the website properties in the algorithm.
3. We justified the use of functionality-supporting components, selected the appropriate framework and demonstrated its applicability in the EA.

Our further plans include training user behavior models to be employed in the EA's fitness function, performing the genetic optimization to produce the final solutions and assessing their quality with representative of the target users.

Acknowledgement. The reported study was funded by Russian Ministry of Education and Science, according to the research project No. 2.2327.2017/4.6, and by RFBR according to the research project No. 16-37-60060 mol_a_dk.

References

1. Anderson, M.R. et al.: Brainwash: A Data System for Feature Engineering. In CIDR (2013).
2. Bakaev, M.: Assessing similarity for case-based web user interface design. Communications in Computer and Information Science (International Conference on Digital Transformation and Global Society), 858, 353-365. Springer, Cham (2018a).
3. Bakaev, M., Gaedke, M.: Application of evolutionary algorithms in interaction design: from requirements and ontology to optimized web interface. In: IEEE Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW 2016), 129-134. (2016).
4. Bakaev, M., Heil, S., Khvorostov, V., Gaedke, M.: HCI Vision for Automated Analysis and Mining of Web User Interfaces. Lecture Notes in Computer Science (International Conference of Web Engineering), 10845, 136-144. Springer, Cham (2018b).
5. Bakaev, M., Khvorostov, V.: Component-based engineering of web user interface designs for evolutionary optimization. In: 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 335-340. IEEE (2018c).
6. Bakaev, M., Khvorostov, V., Heil, S., Gaedke, M.: Web Intelligence Linked Open Data for Website Design Reuse. Lecture Notes in Computer Science (International Conference of Web Engineering), 10360, 370-377. Springer, Cham (2017b)
7. Bakaev, M., Khvorostov, V., Laricheva, T.: Assessing Subjective Quality of Web Interaction with Neural Network as Context of Use Model. Communications in Computer and Information Science (International Conference on Digital Transformation and Global Society), 745, 185-195. Springer, Cham (2017b).
8. Coello, C.A.C.: Multi-objective evolutionary algorithms in real-world applications: Some recent results and current challenges. In: Advances in evolutionary and deterministic

methods for design, optimization and control in engineering and sciences, 3-18. Springer, Cham (2015).
9. De Mantaras et al.: Retrieval, reuse, revision and retention in case-based reasoning. The Knowledge Engineering Review, 20(3), 215-240 (2005).
10. De Renzis, A., Garriga, M., Flores, A., Cechich, A., Zunino, A.: Case-based reasoning for web service discovery and selection. Electronic Notes in Theoretical Computer Science, 321, 89-112 (2016).
11. Freitas, A.A.: Data mining and knowledge discovery with evolutionary algorithms. Springer Science & Business Media (2013).
12. Gajos, K., Wu, A., Weld, D.S.: Cross-device consistency in automatically generated user interfaces. In: 2nd Workshop on Multi-User and Ubiquitous User Interfaces, 7-8 (2005).
13. Glass, R.L.: Facts and fallacies of software engineering. Addison-Wesley Professional (2002).
14. Guo, F., Liu, W.L., Cao, Y., Liu, F.T., Li, M.L.: Optimization design of a webpage based on Kansei engineering. Human Factors and Ergonomics in Manufacturing & Service Industries, 26(1), 110-126 (2016).
15. Ivory, M.Y., Hearst, M.A.: Statistical profiles of highly-rated web sites. In: ACM SIGCHI conference on Human factors in computing systems (CHI), 367-374 (2002).
16. Kazimipour, B., Li, X., Qin, A.K.: A review of population initialization techniques for evolutionary algorithms. In: IEEE Congress on Evolutionary Computation (CEC), 2585-2592 (2014).
17. Kumar, R. et al.: Webzeitgeist: design mining the web. In Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI), 3083-3092 (2013).
18. Marir, F.: Case-based reasoning for an adaptive web user interface. In The International Conference on Computing, Networking and Digital Technologies (ICCNDT2012), 306-315. The Society of Digital Information and Wireless Communication (2012).
19. Martinie, C., Navarre, D., Palanque, P., Fayollas, C.: A generic tool-supported framework for coupling task models and interactive applications. In: 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS), 244-253 (2015).
20. Michalewicz, Z., Hartley, S.J.: Genetic algorithms+ data structures= evolution programs. Mathematical Intelligencer, 18(3), 71 (1996).
21. Oulasvirta, A.: User interface design with combinatorial optimization. Computer, 50(1), 40-47 (2017).
22. Qu, Q. X.: Kansei knowledge extraction based on evolutionary genetic algorithm: an application to e-commerce web appearance design. Theoretical Issues in Ergonomics Science, 16(3), 299-313 (2015).
23. Rocha, R.G., Azevedo, R.R., Sousa, Y.C., Tavares, E.D.A., Meira, S.: A case-based reasoning system to support the global software development. Procedia Computer Science, 35, 194-202 (2014).

Causality-Based Testing in Time Petri Nets*

Elena Bozhenkova[1], Irina Virbitskaite[1,2] and Louchka Popova-Zeugmann[3]

[1] A.P. Ershov Institute of Informatics Systems, SB RAS
6, Acad. Lavrentiev avenue, 630090, Novosibirsk, Russia
Novosibirsk State University
[2] 2, Pirogova st., Novosibirsk, 630090, Russia
[3] Humboldt University of Berlin
Unter den Linden 6, 10099 Berlin, Germany

Abstract. The intention of the paper is towards a causality-based framework for developing, studying, and comparing testing equivalences with causal net and causal tree semantics in the setting of time Petri nets (elementary net systems whose transitions are labeled with time firing intervals, can fire only if their lower time bounds are attained, and are forced to fire when their upper time bounds are reached). We establish the relationships between the equivalences showing the similarity of the semantics under consideration. This allows studying in detail the timing behaviour in addition to the degrees of relative concurrency of processes generated during the functioning of time Petri nets.

1 Introduction

The concept of testing equivalence was put forward by Hennessy and de Nicola in [13]. Two processes are considered testing equivalent if there is no test that can distinguish them. A test is usually itself a process applied to a process by computing both together in parallel. A particular computation is considered to be successful if the test reaches a designated successful state, and the process guarantees the test if every computation is successful. This notion is intuitively appealing and has led to a well-developed mathematical theory of processes that ties together the equivalences and preorders. Testing equivalences were thoroughly investigated and well-understood in the setting of transition systems (see [12, 8] among others) which are a representative of the interleaving approach where concurrency relation is reduced to nondeterminism by treating concurrent execution of actions as the choice between sequentializations of their atomic actions. To overcome the limits of interleaving semantics, concurrency is often modeled by absence of causal dependencies, represented by partial orders, between systems' events. Several well-known variants of partial order testing [2, 15] appeared in the literature. Furthermore, testing equivalences based on causal tree semantics are actively treated as well. Here, the behaviour of a system is represented in the form of a tree with edges labeled by actions and their predecessors. So, information about causality relation between actions is kept precisely.

* This work is supported in part by DFG (project CAVER, grant BE 1267/14-1).

The relationships between causal tree and partial order semantics have been thoroughly studied in [1, 11, 15].

As safety-critical applications often require verification of real time characteristics, in addition to functional or qualitative temporal properties, testing equivalences are expanded in concurrent models with time. The papers [9] and [17] provided an alternative characterization of timed testing for discrete-time transition models, on the base of a notion similar to that of an acceptance set in the testing theory. In [19], decidability questions of interleaving time-sensitive testing are investigated. Semantic theories based on testing were developed for process algebras with timing constraints in the papers [16] and [10]. Both the papers provide alternative characterizations of testing preorders in terms of refusal traces.Also, the authors of [10] proved the possibility of discretization in the context of their algebra and, as a consequence, reduction of dense-timed testing to discrete-timed one. In [7], the testing relations and the results on their alternative characterizations and the possibility of discretization were extended to Petri nets with associating time intervals to arcs from places to transitions. At the same time, to the best of our knowledge, there are only few mentions of a fusion of timing and causality-based semantics, in testing scenario. In this regard, the work [18] is a welcome exception, where time-sensitive testing were treated in the setting of event structures with time characteristics. Also, our origin is the papers [4–6] which contribute to the classification of the wealth of observational equivalences of linear time – branching time spectrum, based on interleaving, causal tree and partial order semantics of dense-time event structures with and without internal actions.

The intention of the paper is towards a causality-based framework for developing, studying, and comparing testing equivalences with causal net and causal tree semantics in the setting of time Petri nets (elementary net systems whose transitions are labeled with time firing intervals, can fire only if their lower time bounds are attained, and are forced to fire when their upper time bounds are reached). We establish the relationships between the equivalences showing the similarity of the semantics under consideration. To do this, we heavily rely on the concept of causal net processes of a time Petri net, which were put forward in the paper [3].

2 Time Petri Nets: Syntax and Interleaving Semantics

In this section, some terminology concerning the model of Petri nets with timing constraints (time intervals on the firings of transitions) and its interleaving semantics in terms of firing sequences are defined.

We start with recalling the definitions of the structure and and behavior of Petri nets (elementary net systems) [14]. We use *Act* to denote an alphabet of actions.

Definition 1. – *A* (labeled over *Act*) Petri net *is a tuple* $\mathcal{N} = (P, T, F, M_0, L)$*, where* P *is a finite set of places and* T *is a finite set of transitions such that* $P \cap T = \emptyset$ *and* $P \cup T \neq \emptyset$*,* $F \subseteq (P \times T) \cup (T \times P)$ *is a flow relation,* $\emptyset \neq M_0 \subseteq P$ *is an initial marking,* $L : T \to Act$ *is a labeling function. For*

$x \in P \cup T$, *let* ${}^\bullet x = \{y \mid (y,x) \in F\}$ *and* $x^\bullet = \{y \mid (x,y) \in F\}$ *be the* preset *and* postset *of* x, *respectively. For* $X \subseteq P \cup T$, *define* ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ *and* $X^\bullet = \bigcup_{x \in X} x^\bullet$.

– *A* marking M *of a Petri net* $\mathcal{N}$ *is any subset of* P. *A transition* $t \in T$ *is* enabled at a marking M *if* ${}^\bullet t \subseteq M$[4]. *Let* $En(M)$ *be the set of transitions enabled at* M.
The firing *of a transition* t *enabled at a marking* M leads *to the new marking* M' *(denoted* $M \xrightarrow{t} M'$*) iff* $M' = (M \setminus {}^\bullet t) \cup t^\bullet$. *We write* $M \xrightarrow{\vartheta} M'$ *iff* $\vartheta = t_1 \dots t_k$ *and* $M = M^0 \xrightarrow{t_1} M^1 \dots M^{k-1} \xrightarrow{t_k} M^k = M'$ *(*$k \geq 0$*). In this case,* ϑ *is a* firing sequence of $\mathcal{N}$ from M (to M'), *and* M' *is a* reachable marking of $\mathcal{N}$ from M. *Let* $RM(\mathcal{N})$ *be the set of all reachable markings of* $\mathcal{N}$ *from* M_0.
We call $\mathcal{N}$ T-restricted *iff* ${}^\bullet t \neq \emptyset \neq t^\bullet$, *for all transitions* $t \in T$; contact-free *iff whenever* t *is a transition enabled at a marking* M, *then* $M \cap t^\bullet = \emptyset$, *for all* $M \in RM(\mathcal{N})$.

Following the approach of [3], we extend the above model to time Petri nets. Let the domain $\mathbb{T}$ of time values be the set of rational numbers. We denote by $[\tau_1, \tau_2]$ the closed interval between two time values $\tau_1, \tau_2 \in \mathbb{T}$. Infinity is allowed at the upper bounds of intervals. Let $Interv$ be the set of all such intervals.

Definition 2. – *A* (labeled over *Act*) time Petri net *is a pair* $\mathcal{TN} = (\mathcal{N}, D)$, *where* $\mathcal{N}$ *is the underlying (labeled over Act) Petri net and* $D : T \to Interv$ *is a* static timing function *associating with each transition a time interval. For a transition* $t \in T$, *the boundaries of the interval* $D(t) \in Interv$ *are called the earliest firing time* Eft *and latest firing time* Lft *of* t.

– *A* state *of* $\mathcal{TN}$ *is a triple* (M, I), *where* M *is a marking and* $I : En(M) \longrightarrow \mathbb{T}$ *is a* dynamic timing function. *The* initial state *of* $\mathcal{TN}$ *is a triple* $S_0 = (M_0, I_0)$, *where* M_0 *is the initial marking and* $I_0(t) = 0$, *for all* $t \in En(M_0)$.
A transition t *enabled at a marking* M *is* fireable from a state $S = (M, I)$ after a delay time $\theta \in \mathbb{T}$ *if* $(Eft(t) \leq I(t) + \theta)$ *and* $(I(t') + \theta \leq Lft(t'))$, *for all* $t' \in En(M))$.
The firing *of a transition* t *fireable from a state* $S = (M, I)$ *after a delay time* θ leads *to the new state* $S' = (M', I')$ *(denoted* $S \xrightarrow{(t,\theta)} S'$*) given as follows:*
(*a*) $M \xrightarrow{t} M'$,
(*b*) $\forall t' \in T \circ I'(t') = \begin{cases} I(t') + \theta, & \text{if } t' \in En(M \setminus {}^\bullet t), \\ 0, & \text{if } t' \in En(M') \setminus En(M \setminus {}^\bullet t), \\ \text{undefined, otherwise.} \end{cases}$
Then, we write $S \xrightarrow{(a,\theta)} S'$, *if* $a = L(t)$. *We use the notation* $S \xrightarrow{\sigma} S'$ *iff* $\sigma = (t_1, \theta_1) \dots (t_k, \theta_k)$ *and* $S = S^0 \xrightarrow{(t_1,\theta_1)} S^1 \dots S^{k-1} \xrightarrow{(t_k,\theta_k)} S^k = S'$ *(*$k \geq 0$*). In this case,* σ *is a* firing sequence of $\mathcal{TN}$ from S (to S'), *and*

[4] For technical convenience, we do not use the classical definition: a transition $t \in T$ is *enabled at a marking* M if ${}^\bullet t \subseteq M$ and $M \cap t^\bullet = \emptyset$. We will require the second part in the definition of the contact-free property.

S' is a reachable state of $\mathcal{TN}$ from S. *Let $\mathcal{FS}(\mathcal{TN})$ be the set of all firing sequences of $\mathcal{TN}$ from S_0, and $RS(\mathcal{TN})$ be the set of all reachable states of $\mathcal{TN}$ from S_0.*
We call $\mathcal{TN}$ T-restricted *iff the underlying Petri net is T-restricted;* contact-free *iff whenever t is a transition fireable from the state $S = (M, I)$ after some delay time θ, then $(M \setminus {}^\bullet t) \cap t^\bullet = \emptyset$, for all $S \in RS(\mathcal{TN})$* [5]*;* time-progressive *iff for all sets $\{t_1, t_2, \ldots, t_n\} \subseteq T$ such that $t_i^\bullet \cap {}^\bullet t_{i+1} \neq \emptyset$ $(1 \leq i < n)$ and $t_n^\bullet \cap {}^\bullet t_1 \neq \emptyset$, it holds that $\sum_{1 \leq i \leq n} Eft(t_i) > 0$*[6]*. In what follows, we will consider only T-restricted, contact-free and time-progressive time Petri nets and denote their class as $\mathbb{TN}$.*

Example 1. A (labeled over $Act = \{a, b, c\}$) time Petri net $\mathcal{TN}$ is shown in Figure 1. Here, the names are depicted near the elements, the flow relation is drawn by the arcs, the initial marking is represented as the set of the places with tokens, and the values of the labeling and timing functions are printed next to the transitions. It is not difficult to check that t_1 and t_3 are transitions enabled at the initial marking M_0 and, moreover, transitions fireable from the initial state $S_0 = (M_0, I_0)$ after a time delay $\theta \in [2, 3]$, where $M_0 = \{p_1, p_2\}$, $I_0(t) = \begin{cases} 0, & \text{if } t \in \{t_1, t_3\}, \\ undefined, & \text{otherwise.} \end{cases}$ The sequence $\sigma = (t_1, 3)\ (t_3, 0)\ (t_2, 2)\ (t_3, 2)\ (t_1, 0)\ (t_5, 2)\ (t_4, 0)$ is a firing sequences of $\mathcal{TN}$ from S_0. Furthermore, it is easy to see that $\mathcal{TN}$ is really T-restricted, contact-free and time-progressive.

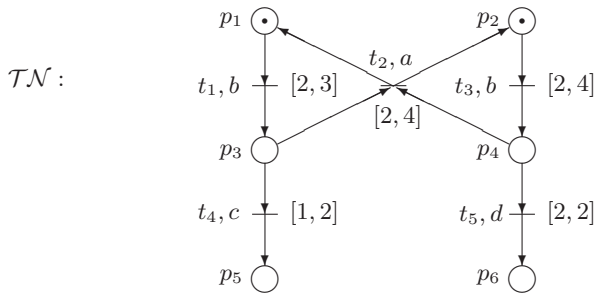


Fig. 1.

3 Causality-Based Semantics of Time Petri Nets

3.1 Preliminaries

We start with considering definitions related to time causal nets.

[5] Clearly, if the underlying Petri net of $\mathcal{TN}$ is contact-free, then $\mathcal{TN}$ must be contact-free as well, but not vice versa.

[6] The time-progressive property shall guarantee the correctness of the modified definition of the contact-free property, for our purposes.

Definition 3. *A* (labeled over *Act*) time net *is a finite, acyclic net* $TN = (B, E, G, l, \tau)$ *with a set* B *of conditions, a set* E *of events, a flow relation* $G \subseteq (B \times E) \cup (E \times B)$ *such that* $\{e \mid (e,b) \in G\} = \{e \mid (b,e) \in G\} = E$, *a labeling function* $l : E \to Act$, *and a time function* $\tau : E \to \mathbb{T}$ *such that* $e\ G^+\ e' \Rightarrow \tau(e) \leq \tau(e')$.

Introduce additional notions and notations for a time net $TN = (B, E, G, l, \tau)$. Let $\prec = G^+$, $\preceq = G^*$, and $\tau(TN) = \max\{\tau(e) \mid e \in E\}$. Specify ${}^\bullet x = \{y \mid (y,x) \in G\}$ and $x^\bullet = \{y \mid (x,y) \in G\}$, for $x \in B \cup E$, and, moreover, ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet = \bigcup_{x \in X} x^\bullet$, for $X \subseteq B \cup E$. Furthermore, define the sets ${}^\bullet TN = \{b \in B \mid {}^\bullet b = \emptyset\}$ and $TN^\bullet = \{b \in B \mid b^\bullet = \emptyset\}$. TN is called a *time causal net*, if $|{}^\bullet b| \leq 1$ and $|b^\bullet| \leq 1$, for all $b \in B$. Notice that $\eta(TN) = (E_{TN}, \preceq_{TN} \cap (E_{TN} \times E_{TN}), l_{TN}, \tau_{TN})$ is a time poset [7]. Given a time causal net over *Act*, $TN = (B,\ E,\ G,\ l,\ \tau)$, $e, e' \in E$, $x, x' \in (B \cup E)$, and $E' \subseteq E$,

- $\downarrow e = \{x \mid x \preceq e\}$ (predecessors), $Earlier(e) = \{e' \in E \mid \tau(e') < \tau(e)\}$ (time predecessors), $x \smile x' \iff \neg((x \prec x') \vee (x' \prec x))$ (concurrency);
- E' is a *downward-closed subset of* E if $\downarrow e' \cap E \subseteq E'$, for all $e' \in E'$. In this case, $Cut(E') = (E'^\bullet \cup {}^\bullet TN) \setminus {}^\bullet E'$. Also, E' is called *timely sound subset of* E if $\tau(e') \leq \tau(e)$, for all $e' \in E'$ and $e \in E \setminus E'$;
- a sequence $\rho = e_1 \dots e_k$ $(k \geq 0)$ of events of TN is a *linearization of* TN if every event of TN appears in the sequence exactly once, and the following holds: $(e_i \preceq e_j \vee \tau(e_i) \leq \tau(e_j)) \Rightarrow i < j$, for all $1 \leq i, j \leq k$. For a linearization $\rho = e_1 \dots e_k$ of TN, define $E_\rho^l = \bigcup_{1 \leq i \leq l} e_i$ $(0 \leq l \leq k)$. Clearly, E_ρ^l is a downward-closed and timely sound subset of E, and, moreover, $\tau(e_k) = \tau(TN)$.

Lemma 1. *Every time causal net* TN *has a linearization* $\rho = e_1 \dots e_k$. *Moreover,* $Cut(E_\rho^l) = \left(Cut(E_\rho^{l-1}) \setminus {}^\bullet e_l\right) \cup e^\bullet{}_l$, *and* $\left(Cut(E_\rho^{l-1}) \setminus {}^\bullet e_l\right) \cap e_l^\bullet = \emptyset$ $(1 \leq l \leq k)$.

Time causal nets, $TN = (B,\ E,\ G,\ l,\ \tau)$ and $TN' = (B',\ E',\ G',\ l',\ \tau')$, are *isomorphic* (denoted $TN \simeq TN'$) iff there exists a bijective mapping $\beta : B \cup E \to B' \cup E'$ such that (i) $\beta(B) = B'$ and $\beta(E) = E'$; (ii) $x\ G\ y \iff \beta(x)\ G'\ \beta(y)$, for all $x, y \in B \cup E$; (iii) $l(e) = l'(\beta(e))$ and $\tau(e) = \tau'(\beta(e))$, for all $e \in E$. We say that TN is a *direct prefix of* TN' (denoted $TN \longrightarrow TN'$) if $B \subseteq B'$, E is a downward-closed and timely sound subset of E', $E' \setminus E = \{e\}$, $\downarrow e \cap E' \subseteq E$, $G = G' \cap (B \times E \cup E \times B)$, $l = l'\,|_E$, and $\tau = \tau'\,|_E$.

3.2 Time Causal Net Semantics

In this subsection, the concept of causality-based net processes of time Petri nets proposed in [3] is considered and studied.

[7] A *(labeled over Act) time poset (partially ordered set)* is a tuple $\eta = (X, \preceq, \lambda, \tau)$ consisting of a finite set X of elements; a reflexive, antisymmetric and transitive relation $\preceq$; a labeling function $\lambda : X \to Act$; and a timing function $\tau : X \to \mathbb{T}$ such that $e \preceq e' \Rightarrow \tau(e) \leq \tau(e')$. Let $\tau(\eta) = \max\{\tau(x) \mid x \in X\}$.

Definition 4. *Given a time Petri net* $\mathcal{TN} = ((P, T, F, M_0, L), D)$ *and a time causal net* $TN = (B, E, G, l, \tau)$,

- *a mapping* $\varphi : B \cup E \to P \cup T$ *is a* homomorphism *from* TN *to* $\mathcal{TN}$ *iff the following conditions hold:*
  - $\varphi(B) \subseteq P$, $\varphi(E) \subseteq T$;
  - *the restriction of* φ *to* ${}^\bullet e$ *is a bijection between* ${}^\bullet e$ *and* ${}^\bullet\varphi(e)$ *and the restriction of* φ *to* $e^\bullet$ *is a bijection between* $e^\bullet$ *and* $\varphi(e)^\bullet$, *for all* $e \in E$;
  - *the restriction of* φ *to* ${}^\bullet TN$ *is a bijection between* ${}^\bullet TN$ *and* M_0;
  - $l(e) = L(\varphi(e))$, *for all* $e \in E$.
- *a pair* $\pi = (TN, \varphi)$ *is a* time process of a time Petri net $\mathcal{TN}$ *iff* TN *is a time causal net and* φ *is a homomorphism from* TN *to* $\mathcal{TN}$.

Given a time process $\pi = (TN, \varphi)$ of $\mathcal{TN}$, a subset $B' \subseteq B_{TN}$, and a transition $t \in En(\varphi(B'))$, the time of enabling (TOE) of t, i.e. the latest global time moment when tokens appear in all input places of t, is defined in [3] as follows: $\mathbf{TOE}_\pi(B', t) = \max\Big(\{\tau_{TN}({}^\bullet b) \mid b \in B'_{[t]} \setminus {}^\bullet TN\} \cup \{0\}\Big)$, where $B'_{[t]} = \{b \in B' \mid \varphi_{TN}(b) \in {}^\bullet t\}$.

Next, define the notion of a *correct time process of* $\mathcal{TN}$.

Definition 5. *A time process* $\pi = (TN, \varphi)$ *of* $\mathcal{TN}$ *is* correct *iff for all* $e \in E$ *it holds:*

(a) $\tau(e) \geq \boldsymbol{TOE}_\pi({}^\bullet e, \varphi(e)) + Eft(\varphi(e))$,
(b) $\forall t \in En(\varphi(C_e)) \circ \tau(e) \leq \boldsymbol{TOE}_\pi(C_e, t) + Lft(t)$, *where* $C_e = Cut(Earlier(e))$.

Let $\mathcal{CP}(\mathcal{TN})$ *denote the set of correct time processes of* $\mathcal{TN}$.

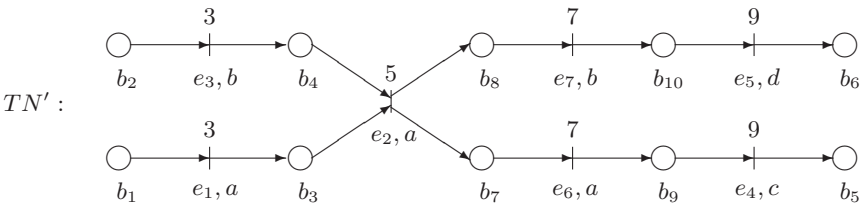


Fig. 2.

Example 2. The time causal net $TN' = (B', E', G', l', \tau')$ is depicted in Figure 2, where the net elements are accompanied by their names, and the values of the functions l' and τ' are indicated nearby the events. Define the time causal nets $TN = (B, E, G, l, \tau)$, with $B = \{b_1, \ldots, b_4\}$, $E = \{e_1, e_3\}$, $G = G' \cap (B \times E \cup E \times B)\}$, $l = l' \mid_E$, $\tau = \tau' \mid_E$. It is easy to see that TN is a prefix of TN'.

Define a mapping φ' from the time causal net TN' (see Figure 2) to the time Petri net $\mathcal{TN}$ (see Figure 1) as follows: $\varphi'(b_i) = p_i$ $(1 \leq i \leq 6)$, $\varphi'(b_i) = p_{i-6}$ $(7 \leq i \leq 10)$, and $\varphi'(e_i) = t_i$ $(1 \leq i \leq 5)$, $\varphi'(e_6) = t_1$, $\varphi'(e_7) = t_3$. Next, for the

time causal net TN, set $\varphi = \varphi' \mid_{E \cup B}$. It is easy to see that $\pi' = (TN', \varphi')$ and $\pi = (TN, \varphi)$ are time processes of $\mathcal{TN}$.

Take $\widetilde{B} = \{b_3, b_4\} \subset B'$, and $t_2 \in En(\varphi'(\widetilde{B}))$. Calculate $\mathbf{TOE}_{\pi'}(\widetilde{B}, t_2) = \max\Big(\{\tau_{TN'}(^\bullet b) \mid b \in \widetilde{B}_{[t_2]} \setminus {}^\bullet TN'\} \cup \{0\}\Big) = \max\Big(\{\tau'(e_1) = 3, \tau'(e_3) = 3\} \cup \{0\}\Big) = 3$. It is not difficult to check that $\pi' = (TN', \varphi')$, $\pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN})$.

We now intend to realize for a time Petri net the relationships between its firing sequences and its correct time processes. For $\pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN}, S)$, define the function FS_π that maps any linearization $\rho = e_1 \ldots e_k$ of TN to the sequence of the form: $FS_\pi(\rho) = (\varphi(e_1), \tau(e_1) - 0) \ \ldots \ (\varphi(e_k), \tau(e_k) - \tau(e_{k-1}))$. The following is a slight modification of Theorems 19, 21 and 22 from [3].

Proposition 1. *(i) Given $\pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN})$ and a linearization ρ of TN, there is a unique firing sequence $FS_\pi(\rho) \in \mathcal{FS}(\mathcal{TN})$.*

(ii) Given $\sigma \in \mathcal{FS}(\mathcal{TN})$ of $\mathcal{TN}$, there is a unique (up to an isomorphism) time process $\pi_\sigma = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN})$ with a unique linearization ρ_σ of TN such that $FS_{\pi_\sigma}(\rho_\sigma) = \sigma$.

From now on, for $\pi = (TN, \varphi)$, $\pi' = (TN', \varphi') \in \mathcal{CP}(\mathcal{TN})$, we write $\pi \longrightarrow \pi'$ in $\mathcal{TN}$ iff $TN \longrightarrow TN'$ and $\varphi = \varphi'|_{B \cup E}$.

Lemma 2. *Given $\sigma \in \mathcal{FS}(\mathcal{TN})$ and $\pi \in \mathcal{CP}(\mathcal{TN})$ such that $\sigma = FS_\pi(\rho)$, where ρ is a linearization of TN_π,*

(i) if $\sigma\widetilde{\sigma} \in \mathcal{FS}(\mathcal{TN})$, then there is $\widetilde{\pi} \in \mathcal{CP}(\mathcal{TN})$ such that $\pi \to \widetilde{\pi}$ in $\mathcal{TN}$ and $\sigma\widetilde{\sigma} = FS_{\widetilde{\pi}}(\rho\widetilde{\rho})$, where $\rho\widetilde{\rho}$ is a linearization of $TN_{\widetilde{\pi}}$;

(ii) if $\pi \to \widetilde{\pi}$ in $\mathcal{TN}$, then there is $\sigma\widetilde{\sigma} \in \mathcal{FS}(\mathcal{TN})$ such that $\sigma\widetilde{\sigma} = FS_{\widetilde{\pi}}(\rho\widetilde{\rho})$, where $\rho\widetilde{\rho}$ is a linearization of $TN_{\widetilde{\pi}}$.

3.3 Time Causal Tree Semantics

Causal trees [11] are synchronisation trees which carry in their labels additional information about causes of actions thus providing us with an interleaving description of concurrent processes which faithfully expresses causality. Time causal trees are an extension of causal trees by adding timing. In the time causal tree of $\mathcal{TN}$, the nodes are simply the firing sequences from $\mathcal{FS}(\mathcal{TN})$, and an arc exists between the two nodes if the second one is an extension of the first one. The causes in the labels of the arc have to be computed from the causality relation of the corresponding time processes of $\mathcal{TN}$.

Definition 6. *The* time causal tree *of $\mathcal{TN}$, $TCT(\mathcal{TN})$, is a tree $(\mathcal{FS}(\mathcal{TN}), A, \phi)$, where $\mathcal{FS}(\mathcal{TN})$ is the set of nodes with the root ϵ; $A = \{(\sigma, \sigma(t,\theta)) \mid \sigma, \sigma(t,\theta) \in \mathcal{FS}(\mathcal{TN})\}$ is the set of arcs; ϕ is the labeling function such that $\phi(\epsilon) = \epsilon$ and $\phi(\sigma, \sigma(t,\theta)) = (l_{\mathcal{TN}}(t), \theta, K)$, where $K = \{n - l + 1 \mid \sigma(t,\theta) = FS_{\pi_{\sigma(t,\theta)}}(e_1 \ \ldots \ e_n \ e)$, for the linearization $e_1 \ \ldots \ e_n \ e$ of $TN_{\pi_{\sigma(t,\theta)}}$, and $e_l \prec_{TN_{\pi_{\sigma(t,\theta)}}} e\}$. Let $path(\sigma)$ be the path starting from the root and finishing in the node σ of $TCT(\mathcal{TN})$*[8].

[8] We assume $path(\epsilon) = \epsilon$. Notice that in $TCT(\mathcal{TN})$, for any node $\sigma \in \mathcal{FS}(\mathcal{TN})$, there is a path starting from the root and finishing in σ.

Example 3. Consider the time Petri net $\mathcal{TN}$ (see Figure 1) and its firing sequence $\sigma = (t_1, 3)\ (t_3, 0)\ (t_2, 2)\ (t_3, 2)\ (t_1, 0)\ (t_5, 2)\ (t_4, 0) \in \mathcal{FS}(\mathcal{TN})$. It is easy to get that $\phi(path(\sigma)) = (a, 3, \emptyset)\ (b, 0, \emptyset)\ (a, 2, \{1, 2\})\ (b, 2, \{1, 2, 3\})\ (a, 0, \{2, 3, 4\})\ (d, 2, \{2, 3, 4, 5\})\ (c, 0, \{2, 4, 5, 6\})$.

We finally establish some relationships between correct time processes and labeled paths in the time causal trees of time Petri nets.

Proposition 2. *(i) Given $\pi \in \mathcal{CP}(\mathcal{TN})$ and $\pi' \in \mathcal{CP}(\mathcal{TN}')$ with an isomorphism $f : \eta(TN_\pi) \to \eta(TN_{\pi'})$, $\phi(path(FS_\pi(\rho))) = \phi'(path(FS_{\pi'}(f(\rho))))$, for any linearization ρ of TN_π.*

(ii) Given $\sigma \in \mathcal{FS}(\mathcal{TN})$ and $\sigma' \in \mathcal{FS}(\mathcal{TN}')$ such that $\phi(path(\sigma)) = \phi'(path(\sigma'))$, there is an isomorphism $f : \eta(TN_{\pi_\sigma}) \to \eta(TN_{\pi_{\sigma'}})$ such that $f(\rho_\sigma) = \rho_{\sigma'}$.

4 Testing Equivalences in Causality-Based Semantics

A kind of causal testing on event structure models has already been defined by Aceto, De Nicola and Fantechi in [2]. Their idea is that the experiments on event structures are pomsets instead of words and the behaviour which is tested for after the experiment consists of a set of actions. Instead of sets of actions, the authors of [15] have used sets of direct extensions of the executed pomsets, as tests. Also, in [15] a stronger version of causal testing has been put forward, based on posets rather than pomsets. Following this approach, we define poset testing equivalence on time Petri nets, relying on their correct time processes. Let $\mathcal{P}os(\mathcal{TN}) = \{TP \text{ is a time poset} | \ \exists \pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN}),\ TP \simeq \eta(TN)$[9]$\}$.

Definition 7. *Given time Petri nets $\mathcal{TN}$ and $\mathcal{TN}'$,*

- *for a time poset TP and a set* **TP** *of time posets, such that $TP \prec TP'$*[10] *for all $TP' \in$* **TP**, *$\mathcal{TN}$* **after** *TP* $\mathbf{MUST}_{pos}$ **TP** *iff for all $\pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN})$ and for all isomorphisms $f : \eta(TN) \longrightarrow TP$, there exists $TP' \in$* **TP**, *$\pi' = (TN', \varphi') \in \mathcal{CP}(\mathcal{TN})$, and an isomorphism $f' : \eta(TN') \longrightarrow TP'$ such that $\pi \to \pi'$ and $f \subseteq f'$;*
- *$\mathcal{TN}$ and $\mathcal{TN}'$ are* poset testing equivalent *(denoted $\mathcal{TN} \sim_{pos} TN'$) iff for all time posets TP and for all sets* **TP** *of time posets, such that $TP \prec TP'$ for all $TP' \in$* **TP**, *it holds: $\mathcal{TN}$* **after** *TP* $\mathbf{MUST}_{pos}$ **TP**$' \iff \mathcal{TN}'$ **after** *TP* $\mathbf{MUST}_{pos}$ **TP**$'$.*

Example 4. Consider the time Petri nets $\mathcal{TN}_1$, $\mathcal{TN}_2$, and $\mathcal{TN}_3$ depicted in Figure 3. It is easy to see that $\mathcal{TN}_1$ and $\mathcal{TN}_2$ are $\sim_{pos}$–equivalent whereas $\mathcal{TN}_2$ and $\mathcal{TN}_3$ are not. Let's make sure the latter. Define posets $TP = (\{x_1, x_2\}, \preceq, \lambda, \tau)$ and $TP' = (\{x_1, x_2, x_3\}, \preceq', \lambda', \tau')$, where $\preceq = \{(x_i, x_i) \mid 1 \leq i \leq 2\}$, $\lambda(x_i) = b$, $\tau(x_1) = \tau'(x_2) = 0$; $\preceq' = \{(x_i, x_i) \mid 1 \leq i \leq 3\} \cup \{(x_2, x_3)\}$, $\lambda'(x_1) =$

[9] Time posets, $\eta = (X, \preceq, \lambda, \tau)$ and $\eta' = (X', \preceq', \lambda', \tau')$, are *isomorphic* (denoted $\eta \simeq \eta'$) iff there is a bijective mapping $\beta : X \to X'$ such that (i) $x \preceq y \iff \beta(x) \preceq' \beta(y)$, for all $x, y \in X$; (ii) $\lambda(x) = \lambda'(\beta(x))$ and $\tau(x) = \tau'(\beta(x))$, for all $x \in X$.

[10] A time poset η is a *direct prefix of a time poset* η' (denoted $\eta \prec \eta'$) iff $X \subseteq X'$, $X' \setminus X = \{x\}$, $\preceq = \preceq' \cap (X \times X)$, $\lambda = \lambda' \mid_X$, $\tau = \tau' \mid_X$, and x is a maximal w.r.t. $\preceq'$ element of X'.

$\lambda'(x_2) = b$, $\lambda'(x_3) = a$, $\tau'(x_1) = \tau'(x_2) = 0$, and $\tau'(x_3) = 3.9$. For any time process $\pi_2 = (TN_2, \varphi_2) \in \mathcal{CP}(\mathcal{TN}_2)$ with E_{TN_2} consisting of two concurrent events with labels b and time values 0, and any isomorphism $f_2 \colon \eta(TN_2) \longrightarrow TP$, we can find $\pi_2' = (TN_2', \varphi_2') \in \mathcal{CP}(\mathcal{TN}_2)$ with $E_{TN_2'}$ consisting of two concurrent events with labels b and time values 0 and some third event with label a and time value 3.9, which is causally preceded by one of the b's, and an isomorphism $f_2' \colon \eta(TN_2') \longrightarrow TP'$ such that $\pi_2 \to \pi_2'$ and $f_2 \subset f_2'$. But this is not the case in $\mathcal{TN}_3$.

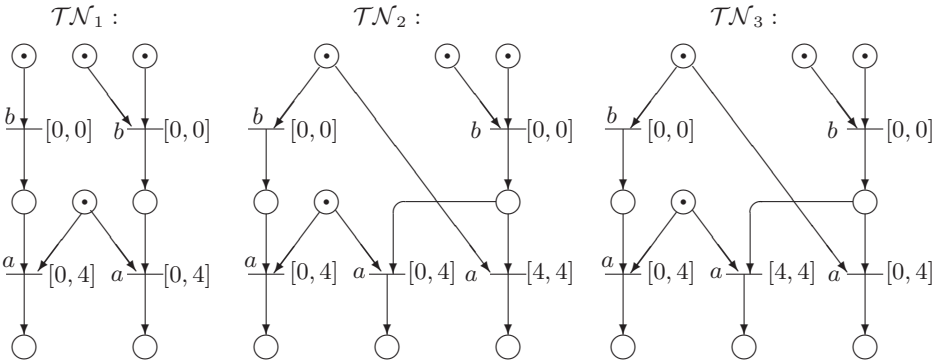


Fig. 3.

Second, the definition of testing equivalence on time causal trees is developed. For this we adapt the concept of causal tree testing on event structures from [15] to time Petri nets, in so doing the experiments and tests are constructed over the alphabet $Act \times \mathbb{T} \times 2^{\mathbb{N}}$ instead of over $Act \times 2^{\mathbb{N}}$. We shall need the set $\mathcal{L}(TCT) = \{\phi(u) \in (Act \times \mathbb{T} \times 2^{\mathbb{N}})^* \mid u \text{ is a path in the time causal tree } TCT \text{ starting from its root}\}$.

Definition 8. *Given time Petri nets $\mathcal{TN}$ and $\mathcal{TN}'$ with their time causal trees $TCT(\mathcal{TN})$ and $TCT(\mathcal{TN}')$, respectively,*

- *for a sequence $w \in (Act \times \mathbb{T} \times 2^{\mathbb{N}})^*$ and a set $\mathbf{W} \subseteq (Act \times \mathbb{T} \times 2^{\mathbb{N}})$, $TCT(\mathcal{TN})$* **after** *w* **MUST W** *iff for all paths u in $TCT(\mathcal{TN})$ from its root to a node n such that $\phi(u) = w$, there exists a label $(a, d, K) \in \mathbf{W}$ and an arc r starting from n such that $\phi(r) = (a, d, K)$;*
- *$TCT(\mathcal{TN})$ and $TCT(\mathcal{TN}')$ are causal tree testing equivalent (denoted $TCT(\mathcal{TN}) \sim_{ct} TCT(\mathcal{TN})$) iff for all $w \in (Act \times \mathbb{T} \times 2^{\mathbb{N}})^*$ and $\mathbf{W} \subseteq (Act \times \mathbb{T} \times 2^{\mathbb{N}})$, $TCT(\mathcal{TN})$* **after** *w* **MUST W** *$\iff$ $TCT(\mathcal{TN}')$* **after** *w* **MUST W**.

Lemma 3. *Given time Petri nets $\mathcal{TN}_1$ and $\mathcal{TN}_2$,*

(i) $\mathcal{P}os(\mathcal{TN}_1) = \mathcal{P}os(\mathcal{TN}_2) \iff \mathcal{L}(TCT(\mathcal{TN}_1)) = \mathcal{L}(TCT(\mathcal{TN}_2))$;
(ii) $\mathcal{TN}_1 \sim_{pos} \mathcal{TN}_2 \Longrightarrow \mathcal{P}os(\mathcal{TN}_1) = \mathcal{P}os(\mathcal{TN}_2)$,
$TCT(\mathcal{TN}_1) \sim_{ct} TCT(\mathcal{TN}_2) \Longrightarrow \mathcal{L}(TCT(\mathcal{TN}_1)) = \mathcal{L}(TCT(\mathcal{TN}_2))$.

We finally establish the coincidence of poset and causal tree testing equivalences, in the setting of time Petri nets. The proof of the result can be found in Appendix.

Theorem 1. *Given time Petri nets $\mathcal{TN}$ and $\mathcal{TN}'$,*

$$\mathcal{TN}_1 \sim_{pos} \mathcal{TN}_2 \iff TCT(\mathcal{TN}_1) \sim_{ct} TCT(\mathcal{TN}_2).$$

5 Concluding Remarks

We have shown that some of the causality-based testing equivalences actively treated in the untimed and timed event structures literature may be lifted to the realm of time Petri nets. In particular, we have defined testing equivalences based on time causal trees and time causal nets, in the setting of safe Petri nets (elementary net systems) with strong timing (transitions are labeled with time firing intervals, enabled transitions are able to fire only if their lower time bounds are attained, and are forced to fire when their upper time bounds are reached). In doing so, we dealt with three behavioral representations of a time Petri net: firing sequences representing interleaving semantics, time causal net processes, from causal nets of which partial orders are derived, and the causal tree semantics constructed from the firing sequences and partial orders. We have realized for a time Petri net the relationships between its firing sequences and correct time processes, on the one hand, and the labeled paths in its time causal tree and correct time processes, on the other hand. As a main result, the coincidence between the testing equivalences in the semantics of time partial orders and time causal trees has been established. It is worth noticing that the result also works for the untimed versions of the equivalences in the setting of untimed contact-free elementary net systems.

As for future work, we plan to see the place of the equivalences and semantics under consideration in the lattice of those in the linear-time - branching-time and interleaving - partial order spectra, constructed in the paper [20]. Also, we intend to extend the results obtained to time Petri nets with invisible actions.

References

1. Aceto L.: History preserving, causal and mixed-ordering equivalence over stable event structures *Fundamenta Informaticae* **17(4)** (1992) 319–331.
2. Aceto L., De Nicola R., Fantechi A.: Testing equivalences for event structures. *Lecture Notes in Computer Science* **280** (1987) 1–20.
3. Aura T., Lilius J.: A causal semantics for time Petri nets. *Theoretical Computer Scinece* **243** (2000) 409–447.
4. Andreeva M., Bozhenkova E., Virbitskaite I.: Analysis of timed concurrent models based on testing equivalenc. *Fundamenta Informaticae* **43** (2000) 1–20.
5. Andreeva M., Virbitskaite I.: Timed equivalences for timed event structures. *Lecture Notes in Computer Science* **3606** (2005) 16–25.
6. Andreeva M., Virbitskaite I.: Observational Equivalences for Timed Stable Event Structures. *Fundamenta Informaticae* **72(1-3)** (2006) 1–19.

7. Bihler E., Vogler W.: Timed Petri Nets: Efficiency of Asynchronous Systems. *Lecture Notes in Computer Science* **3185** (2004) 25–58.
8. Cleaveland R., Hennessy M.: Testing equivalence as a bisimulation equivalence. *Lecture Notes in Computer Science* **407** (1989) 11–23.
9. Cleaveland R., Zwarico A.E.: A theory of testing for real-time. Proc. 6th IEEE Symp. on Logic in Comput. Sci. (*LICS'91*), Amsterdam, The Netherlands (1991) 110–119.
10. Corradini F., Vogler W., Jenner L.: Comparing the Worst-Case Efficiency of Asynchronous Systems with PAFAS. Tech. Rep. N 2000-6, Inst. fur Informatik of Univ. of Augsburg.
11. Darondeau Ph., Degano P.: Refinement of actions in event structures and causal trees. *Theoretical Computer Science* **118** (1993) 21–48.
12. De Nicola R.: Extensional equivalences for transition systems. *Acta Informatica* **24** (1987) 211-237.
13. De Nicola R., Hennessy M.: Testing equivalence for processes. *Theoretical Computer Science* **34** (1984) 83–133.
14. Rozenberg, G., Engelfriet, J.: Elementary Net Systems. *Lecture Notes in Computer Science* **1491** (1998) 12–121.
15. Goltz U., Wehrheim H.: Causal testing. *Lecture Notes in Computer Science* **1113** (1996) 394–406.
16. Hennessy M., Regan T.: A process algebra for timed systems. *Information and Computation* **117** (1995) 221–239.
17. Llana L., de Frutos D.: Denotational semantics for timed testing. *Lecture Notes in Computer Science* **1233** (1997) 368–382.
18. Murphy D.: Time and duration in noninterleaving concurrency. *Fundamenta Informaticae* **19** (1993) 403–416.
19. Steffen B., Weise C.: Deciding testing equivalence for real-time processes with dense time. *Lecture Notes in Computer Science* **711** (1993) 703–713.
20. Virbitskaite I., Bushin D., Best E.: True concurrent equivalences in time Petri nets. *Fundamenta Informaticae* **149(4)** (2016) 401–418.

Appendix

Proof of Theorem 1. Let $TCT(\mathcal{TN}_i) = (\mathcal{FS}(\mathcal{TN}_i), A_i, \phi_i)$ $(i = 1, 2)$.

($\Rightarrow$) Assume $\mathcal{TN}_1 \sim_{pos} \mathcal{TN}_2$. Then, $\mathcal{P}os(\mathcal{TN}_1) = \mathcal{P}os(\mathcal{TN}_2)$, by Lemma 3(ii). Thanks to Lemma 3(i), $\mathcal{L}(TCT(\mathcal{TN}_1)) = \mathcal{L}(TCT(\mathcal{TN}_2))$. We shall show that $TCT(\mathcal{TN}_1) \sim_{ct} TCT(\mathcal{TN}_2)$. Take arbitrary $w \in (Act \times \mathbb{T} \times 2^{\mathbb{N}})^*$ and $\mathbf{W} \subseteq (Act \times \mathbb{T} \times 2^{\mathbb{N}})$. W.l.o.g. let $\mid w \mid = n$ $(n \geq 0)$. Suppose $TCT(\mathcal{TN}_1)$ **after** w **MUST W**, i.e. for all paths u in $TCT(\mathcal{TN}_1)$ from its root to a node σ, such that $\phi_1(u) = w$, there exists a label $(a, \theta, K) \in \mathbf{W}$ and an arc r starting from σ such that $\phi_1(r) = (a, \theta, K)$. Check that $TCT(\mathcal{TN}_2)$ **after** w **MUST W**.

If $w \notin \mathcal{L}(TCT(\mathcal{TN}_1)) = \mathcal{L}(TCT(\mathcal{TN}_2))$, then the result is obvious. Consider the case with $w \in \mathcal{L}(TCT(\mathcal{TN}_1)) = \mathcal{L}(TCT(\mathcal{TN}_2))$. Due to the fact, we can take an arbitrary path u from the root to some node $\sigma \in \mathcal{FS}(\mathcal{TN}_1)$, such that $\phi_1(u) = w$. By Proposition 1(ii), there is a unique (up to an isomorphism) time process $\pi_\sigma = (TN_\sigma, \varphi_\sigma) \in \mathcal{CP}(\mathcal{TN}_1)$ with a unique linearization $\rho_\sigma = e^\sigma_1 \dots e^\sigma_n$ of TN_σ such that $FS_{\pi_\sigma}(\rho_\sigma) = \sigma$. Let $TP_w = \eta(TN_\sigma)$. Clearly, $TP_w \in \mathcal{P}os(\mathcal{TN})$.

For each $(a, \theta, K) \in \mathbf{W}$, construct the time poset $TP_{(a,\theta,K)} = (X, \preceq, \lambda, \tau)$ as follows: $X = E_{TN_\sigma} \cup \{e_{(a,\theta,K)}\}$ $(e_{(a,\theta,K)} \notin E_{TN_\sigma})$; $\preceq = \preceq_{TN_\sigma} \cup \{(e^\sigma_{n-k+1},$

$e_{(a,\theta,K)}) \mid k \in K\}$; $\lambda \mid_{E_{TN_\sigma}} = \lambda_{TN_\sigma}$, $\lambda(e_{(a,\theta,K)}) = a$; $\tau \mid_{E_{TN_\sigma}} = \tau_{TN_\sigma}$, $\tau(e_{(a,\theta,K)}) = \tau(TN_\sigma) + \theta$. Let $\mathbf{TP_W} = \{TP_{(a,\theta,K)} \mid (a,\theta,K) \in \mathbf{W}\}$. By the construction, it holds that $TP_w \prec TP_{(a,\theta,K)}$, for all $TP_{(a,\theta,K)} \in \mathbf{TP_W}$.

Check that $\mathcal{TN}_1$ **after** TP_w $\mathbf{MUST}_{pos}$ $\mathbf{TP_W}$, i.e. for all $\pi_1 = (TN_1, \varphi_1) \in \mathcal{CP}(\mathcal{TN}_1)$ and for all isomorphisms $f_1 : \eta(TN_1) \longrightarrow TP_w$, there exists $TP' \in \mathbf{TP_W}$, $\pi_1' = (TN_1', \varphi_1') \in \mathcal{CP}(\mathcal{TN}_1)$ and an isomorphism $f_1' : \eta(TN_1') \longrightarrow TP'$ such that $\pi_1 \to \pi_1'$ and $f_1 \subseteq f_1'$. Take arbitrary $\pi_1 = (TN_1, \varphi_1) \in \mathcal{CP}(\mathcal{TN}_1)$ and isomorphism $f_1 : \eta(TN_1) \longrightarrow TP_w$. Clearly, $(f_1)^{-1} : \eta(TN_\sigma) \to \eta(TN_1)$ is an isomorphism. Due to Proposition 2(ii), $e_1^1 \dots e_n^1 = \rho_1 = (f_1)^{-1}(\rho_\sigma)$ is a linearization of TN_1 such that $w = \phi(path(\sigma_1 = FS_{\pi_1}(\rho_1)))$. As $TCT(\mathcal{TN}_1)$ **after** w **MUST W**, there exists a label $(a_1', \theta_1', K_1') \in \mathbf{W}$ and an arc r_1 starting from the node σ_1 such that $\phi_1(r_1) = (a_1', \theta_1', K_1')$. Then, we can find $TP_1' = TP_{(a_1',\theta_1',K_1')} \in \mathbf{TP_W}$ and, hence, $\{e_{(a_1',\theta_1',K_1')}\} = E_{TP_1'} \setminus E_{TN_\sigma}$, $a_1' = \lambda_{TP_1'}(e_{(a_1',\theta_1',K_1')})$, $\theta_1' = \tau_{TP_1'}(e_{(a_1',\theta_1',K_1')}) - \tau(TN_\sigma)$, $K_1' = \{n-l+1 \mid e_l^\sigma \preceq_{TP_1'} e_{(a_1',\theta_1',K_1')}\}$, thanks to the construction of $\mathbf{TP_W}$. Moreover, there is $\sigma_1' = \sigma_1(t_1', \theta_1') \in \mathcal{FS}(\mathcal{TN}_1)$, for some $t_1' \in T_{\mathcal{TN}_1}$, such that $r_1 = (\sigma_1, \sigma_1')$ and $\phi_1(\sigma_1', \sigma_1') = (l_{\mathcal{TN}_1}(t_1') = a_1', \theta_1', K_1')$, due to the definition of $TCT(\mathcal{TN}_1)$. By virtue of Lemma 2(i), there is $\pi_1' = (TN_1', \varphi_1') \in \mathcal{CP}(\mathcal{TN}_1)$ such that $\pi_1 \longrightarrow \pi_1'$ and $\sigma_1' = FS_{\pi_1'}(\rho_1')$, for some linearization $\rho_1' = \rho_1\widetilde{\rho}$ of TN_1'. W.l.o.g. suppose $\widetilde{\rho} = e_1'$. Then, $\varphi_1'(e_1') = t_1'$. Specify a mapping $f_1' : \eta(TN_1') \to TP_1'$ as follows: $f_1' \mid_{E_{\eta(TN_1)}} = f_1$, $f_1'(e_1') = e_{(a_1',\theta_1',K_1')}$. Moreover, we have: $\lambda_{\eta(TN_1')}(e_1') = a_1' = \lambda_{TP_1'}(e_{(a_1',\theta_1',K_1')})$; $\tau_{\eta(TN_1')}(e_1') = \theta_1' + \tau(TN_1) = \theta_1' + \tau(TN_\sigma) = \tau_{TP_1'}(e_{(a_1',\theta_1',K_1')})$; $e_{n-k+1}^1 \preceq_{\eta(TN_1')} e_1' \iff f_1'(e_{n-k+1}^1) = e_{n-k+1}^\sigma \preceq_{TP_1'} e_{(a_1',\theta_1',K_1')}$, for all $k \in K_1'$. So, f_1' is an isomorphism and $f_1 \subseteq f_1'$. Thus, $\mathcal{TN}_1$ **after** TP_w $\mathbf{MUST}_{pos}$ $\mathbf{TP_W}$. Hence, $\mathcal{TN}_2$ **after** TP_w $\mathbf{MUST}_{pos}$ $\mathbf{TP_W}$, by the theorem assumption.

We further show that $TCT(\mathcal{TN}_2)$ **after** w **MUST W**, i.e. for all paths u_2 in $TCT(\mathcal{TN}_2)$ from its root to a node σ_2 such that $\phi_2(u_2) = w$, there exists $(a,\theta,K) \in \mathbf{W}$ and an arc r_2 starting from σ_2 such that $\phi_2(r_2) = (a,\theta,K)$. Take an arbitrary path u_2 in $TCT(\mathcal{TN}_2)$ from its root to a node σ_2 such that $\phi_2(u_2) = w$. Since $w \in \mathcal{L}(TCT(\mathcal{TN}_2))$, at least one such u_2 exists in $TCT(\mathcal{TN}_2)$. By Proposition 1(ii), there is a unique (up to an isomorphism) time process $\pi_{\sigma_2} = (TN_2, \varphi_2) \in \mathcal{CP}(\mathcal{TN}_2)$ with a unique linearization $\rho_2 = e_1^2 \dots e_n^2$ of TN_2 such that $FS_{\pi_{\sigma_2}}(\rho_2) = \sigma_2$. By virtue of Proposition 2(i), there exists an isomorphism $f_2 : \eta(TN_2) \longrightarrow TP_w$ such that $f_2(\rho_2) = \rho_\sigma$. Since $\mathcal{TN}_2$ **after** TP_w $\mathbf{MUST}_{pos}$ $\mathbf{TP_W}$, there exists $TP_2' \in \mathbf{TP_W}$, $\pi_{\sigma_2}' = (TN_2', \varphi_2') \in \mathcal{CP}(\mathcal{TN}_2)$ and an isomorphism $f_2' : \eta(TN_2') \longrightarrow TP_2'$, such that $\pi_{\sigma_2} \longrightarrow \pi_2'$ and $f_2 \subseteq f_2'$. Due to Lemma 2(ii), there exists $\sigma_2\widetilde{\sigma_2} \in \mathcal{FS}(\mathcal{TN}_2)$ such that $\sigma_2\widetilde{\sigma_2} = FS_{\pi_2'}(\rho_2\widetilde{\rho_2})$, for some linearization $\rho_2\widetilde{\rho_2}$ of TN_2'. By the construction of $\mathbf{TP_W}$, there is $(a,\theta,K) \in \mathbf{W}$ such that $TP_2' = TP_{(a,\theta,K)}$, and, hence, $\{e_{(a,\theta,K)}\} = E_{TP_2'} \setminus E_{TN_\sigma}$. As $TN_2 \longrightarrow TN_2'$ and $TP_w \prec TP_2'$, we can w.l.o.g. suppose that $\{e_2'\} = E_{TN_2'} \setminus E_{TN_2}$. This implies that $\widetilde{\rho_2} = e_2'$ and $f_2'(e_2') = e_{(a,\theta,K)}$. Furthermore, due to f_2' being an isomorphim, we have: $\lambda_{\eta(TN_2')}(e_2') = \lambda_{TP_2'}(e_{(a,\theta,K)}) = a$, $\tau_{\eta(TN_2')}(e_2') = \tau_{TP_2'}(e_{(a,\theta,K)}) = \tau(TN_\sigma) + \theta = \tau(TN_2) + \theta$, and $e_i^\sigma \preceq_{TP_2'} e_{(a,\theta,K)} \iff (f_2')^{-1}(e_i^\sigma) = e_i^2 \preceq_{\eta(TN_2')} e_2'$, for all $1 \le i \le n$. So, $\widetilde{\sigma_2} = (\varphi_2'(e_2'), \theta)$ and $e_{n-k+1}^2 \preceq_{\eta(TN_2')} e_2'$, for all $k \in K$. Hence, in $TCT(\mathcal{TN}_2)$ there is an arc

$r_2 = (\sigma_2, \sigma_2\widetilde{\sigma_2})$ such that $\phi_2(r_2) = (a, \theta, K)$. Therefore, we get that $TCT(\mathcal{TN}_1)$ **after** w **MUST W** $\Rightarrow TCT(\mathcal{TN}_2)$ **after** w **MUST W**.

By symmetry, we have that $\mathcal{TN}_1 \sim_{pos} \mathcal{TN}_2 \Rightarrow TCT(\mathcal{TN}_1) \sim_{ct} TCT(\mathcal{TN}_2)$.

($\Leftarrow$) Assume $TCT(\mathcal{TN}_1) \sim_{ct} TCT(\mathcal{TN}_2)$. By Lemma 3(ii), $\mathcal{L}(TCT(\mathcal{TN}_1)) = \mathcal{L}(TCT(\mathcal{TN}_2))$. Moreover, $\mathcal{P}os(\mathcal{TN}_1) = \mathcal{P}os(\mathcal{TN}_2)$, thanks to Lemma 3(i). We shall show that $\mathcal{TN}_1 \sim_{pos} \mathcal{TN}_2$. Take an arbitrary time poset TP and arbitrary set **TP** of time posets such that $TP \prec\!\!\!\cdot\, TP'$, for all $TP' \in$ **TP**. Suppose that $\mathcal{TN}_1$ **after** TP **MUST**$_{pos}$ **TP**, i.e. for all $\pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN}_1)$ and for all isomorphisms $f : \eta(TN) \longrightarrow TP$, there exists $TP' \in$ **TP**, $\pi' = (TN', \varphi') \in \mathcal{CP}(\mathcal{TN}_1)$ and an isomorphism $f' : \eta(TN') \longrightarrow TP'$ such that $\pi \to \pi'$ and $f \subseteq f'$. Check that $\mathcal{TN}_2$ **after** TP **MUST**$_{pos}$ **TP**.

In the case when $TP \notin \mathcal{P}os(\mathcal{TN}_1) = \mathcal{P}os(\mathcal{TN}_2)$, the result is obvious. Suppose $TP \in \mathcal{P}os(\mathcal{TN}_1) = \mathcal{P}os(\mathcal{TN}_2)$. Then, we can take an arbitrary $\pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN}_1)$ and arbitrary isomorphism $f : \eta(TN) \longrightarrow TP$. Consider an arbitrary linearization ρ of TN, which exists, due to Lemma 1. By Proposition 1(i), there is a unique firing sequence $\sigma = FS_\pi(\rho) \in \mathcal{FS}(\mathcal{TN}_1)$. By Proposition 1(ii), we can w.l.o.g. assume that $\pi = \pi_\sigma = (TN_\sigma, \varphi_\sigma)$ and $\rho = \rho_\sigma = e^\sigma_1 \dots e^\sigma_n$ ($n \geq 0$). Denote $\phi(path(\sigma))$ as w_{TP}. So, $w_{TP} \in \mathcal{L}(TCT(\mathcal{TN}_1))$.

For each $TP' \in$ **TP**, construct a triple $(a_{TP'}, \theta_{TP'}, K_{TP'}) \in (Act \times \mathbb{T} \times 2^{\mathbb{N}})$ as follows. Let $\{e_{TP'}\} = X_{TP'} \setminus X_{TP}$. Then, we have: $a_{TP'} = \lambda_{TP'}(e_{TP'})$, $\theta_{TP'} = \tau_{TP'}(e_{TP'}) - \tau(TN_\sigma)$, $K_{TP'} = \{n - l + 1 \mid f(e^\sigma_l) \preceq_{TP'} e_{TP'}, 1 \leq l \leq n\}$. Determine the set $\mathbf{W_{TP}} = \{(a_{TP'}, \theta_{TP'}, K_{TP'}) \mid TP' \in \mathbf{TP}\}$.

Check that $TCT(\mathcal{TN}_1)$ **after** w_{TP} **MUST** $\mathbf{W_{TP}}$, i.e. for all paths u_1 in $TCT(\mathcal{TN}_1)$ from its root to a node σ_1 such that $\phi_1(u_1) = w_{TP}$, there exists $(a, \theta, K) \in \mathbf{W_{TP}}$ and an arc r_1 starting from σ_1 such that $\phi_1(r_1) = (a, \theta, K)$. Take an arbitrary path u_1 in $TCT(\mathcal{TN}_1)$ from its root to a node σ_1 such that $\phi_1(u_1) = w_{TP}$. Since $w_{TP} \in \mathcal{L}(TCT(\mathcal{TN}_1))$, at least one such u_1 exists in $TCT(\mathcal{TN}_1)$. By Proposition 1(ii), there is a unique (up to an isomorphism) time process $\pi_{\sigma_1} = (TN_1, \varphi_1) \in \mathcal{CP}(\mathcal{TN}_1)$ with a unique linearization $\rho_{\sigma_1} = e^1_1 \dots e^1_n$ of TN_1 such that $FS_{\pi_{\sigma_1}}(\rho_{\sigma_1}) = \sigma_1$. By virtue of Proposition 2(i), there exists an isomorphism $f_{\sigma_1\sigma} \colon \eta(TN_1) \longrightarrow \eta(TN_\sigma)$, such that $f_{\sigma_1\sigma}(\rho_{\sigma_1}) = \rho_\sigma$. Moreover, we get $f_1 = f \circ f_{\sigma_1\sigma} \colon \eta(TN_1) \longrightarrow TP$. Since $\mathcal{TN}_1$ **after** TP **MUST**$_{pos}$ **TP**, there exists $TP'_1 \in$ **TP**, $\pi'_1 = (TN'_1, \varphi'_1) \in \mathcal{CP}(\mathcal{TN}_1)$ and an isomorphism $f'_1 : \eta(TN'_1) \longrightarrow TP'_1$, such that $\pi_{\sigma_1} \longrightarrow \pi'_1$ and $f_1 \subseteq f'_1$. Due to Lemma 2(ii), there exists $\sigma_1\widetilde{\sigma_1} \in \mathcal{FS}(\mathcal{TN}_1)$ such that $\sigma_1\widetilde{\sigma_1} = FS_{\pi'_1}(\rho_{\sigma_1}\widetilde{\rho_1})$, for some linearization $\rho_{\sigma_1}\widetilde{\rho_1}$ of TN'_1. Also, there is $(a_{TP'_1}, \theta_{TP'_1}, K_{TP'_1}) \in \mathbf{W_{TP}}$, by the construction of $\mathbf{W_{TP}}$, and, hence, $\{e_{TP'_1}\} = X_{TP'_1} \setminus X_{TP}$. As $TN_1 \longrightarrow TN'_1$ and $TP \prec\!\!\!\cdot\, TP'_1$, we can w.l.o.g. suppose that $\{e'_1\} = E_{TN'_1} \setminus E_{TN_1}$, i.e. $\widetilde{\rho_1} = e'_1$, and $f'_1(e'_1) = e_{TP'_1}$. Furthermore, due to f'_1 being an isomorphism, we have: $\lambda_{\eta(TN'_1)}(e'_1) = \lambda_{TP'_1}(e_{TP'_1}) = a_{TP'_1}$, $\tau_{\eta(TN'_1)}(e'_1) = \tau_{TP'_1}(e_{TP'_1}) = \tau(TN_\sigma) + \theta_{TP'_1}$ $= \tau(TN_1) + \theta_{TP'_1}$, and $f(e^\sigma_i) \preceq_{TP'_1} e_{TP'_1} \iff (f'_1)^{-1}(f(e^\sigma_i)) = e^1_i \preceq_{\eta(TN'_1)} e'_1$, for all $1 \leq i \leq n$. So, $e^1_{n-k+1} \preceq_{\eta(TN'_1)} e'_1$, for all $k \in K_{TP'_1}$, and $\widetilde{\sigma_1} = (\varphi'_1(e'_1), \theta_{TP'_1})$. Hence, in $TCT(\mathcal{TN}_1)$ there is an arc $r_1 = (\sigma_1, \sigma_1\widetilde{\sigma_1})$ such that $\phi_1(r_1) = (a_{TP'_1}, \theta_{TP'_1}, K_{TP'_1})$. Therefore, we get that $TCT(\mathcal{TN}_1)$ **after** w_{TP}

MUST $\mathbf{W_{TP}}$. By the theorem assumption, we have that $TCT(\mathcal{TN}_2)$ **after** w_{TP} **MUST** $\mathbf{W_{TP}}$.

We further show that $\mathcal{TN}_2$ **after** TP $\mathbf{MUST}_{pos}$ **TP**, i.e. for all $\pi_2 = (TN_2, \varphi_2) \in \mathcal{CP}(\mathcal{TN}_2)$ and for all isomorphisms $f_2 : \eta(TN_2) \longrightarrow TP$, there exists $TP'_2 \in \mathbf{TP}$, $\pi'_2 = (TN'_2, \varphi'_2) \in \mathcal{CP}(\mathcal{TN}_2)$ and an isomorphism $f'_2 : \eta(TN'_2) \longrightarrow TP'_2$ such that $\pi_2 \to \pi'_2$ and $f_2 \subseteq f'_2$. Take an arbitrary $\pi_2 = (TN_2, \varphi_2) \in \mathcal{CP}(\mathcal{TN}_2)$ and arbitrary isomorphism $f_2 : \eta(TN_2) \longrightarrow TP$. Define the isomorphism $\bar{f}_2 = f_2^{-1} \circ f \colon \eta(TN_\sigma) \longrightarrow \eta(TN_2)$. Due to Proposition 2(ii), $w_{TP} = \phi_2(path(\sigma_2 = FS_{\pi_2}(\rho_2)))$, for the linearization $\bar{f}_2(\rho_\sigma) = \rho_2 = e^2_1 \dots e^2_n$ of TN_2. As $TCT(\mathcal{TN}_2)$ **after** w_{TP} **MUST** $\mathbf{W_{TP}}$, there exists $(a'_2, \theta'_2, K'_2) \in \mathbf{W_{TP}}$ and an arc r_2 starting from σ_2 such that $\phi_2(r_2) = (a'_2, \theta'_2, K'_2)$. Then, we can find $TP'_2 \in \mathbf{TP}$ such that $(a_{TP'_2}, \theta_{TP'_2}, K_{TP'_2}) = (a'_2, \theta'_2, K'_2)$, thanks to the construction of $\mathbf{W_{TP}}$. Moreover, there is $\sigma'_2 = \sigma_2(t'_2, \theta'_2) \in \mathcal{FS}(\mathcal{TN}_2)$, for some $t'_2 \in T_{\mathcal{TN}_2}$, such that $\phi_2(\sigma_2, \sigma_2(t'_2, \theta'_2)) = (l_{\mathcal{TN}_2}(t'_2) = a'_2, \theta'_2, K'_2)$, due to the definition of $TCT(\mathcal{TN}_2)$. By virtue of Lemma 2(i), there is $\pi'_2 = (TN'_2, \varphi'_2) \in \mathcal{CP}(\mathcal{TN}_2)$ such that $\pi_2 \longrightarrow \pi'_2$ and $\sigma'_2 = FS_{\pi'_2}(\rho'_2)$, for some linearization $\rho'_2 = \rho_2\widetilde{\rho}$ of π'_2. W.l.o.g. suppose $\{e'_2\} = E_{TN'_2} \setminus E_{TN_2}$. Then, $\varphi'_2(e'_2) = t'_2$. Specify a mapping $f'_2 : \eta(TN'_2) \to TP'_2$ as follows: $f'_2 \mid_{E_{\eta(TN_2)}} = f_2$, $f'_2(e'_2) = e_{(a'_2, \theta'_2, K'_2)}$. Moreover, we have: $\lambda_{\eta(TN'_2)}(e'_2) = a'_2 = \lambda_{TP'_2}(e_{(a'_2, \theta'_2, K'_2)})$; $\tau_{\eta(TN'_2)}(e'_2) = \theta'_2 + \tau(TN_2) = \theta'_2 + \tau(TN_\sigma) = \tau_{TP'_2}(e_{(a'_2, \theta'_2, K'_2)})$; $e^2_{n-k+1} \preceq_{\eta(TN'_2)} e'_2 \iff f'_2(\bar{f}_2(e^\sigma_{n-k+1})) \preceq_{TP'_2} f'_2(e'_2) \iff f(e^\sigma_{n-k+1}) \preceq_{TP'_2} e_{(a'_2, \theta'_2, K'_2)}$, for all $k \in K'_2$. So, f'_2 is an isomorphism and $f_2 \subseteq f'_2$. Thus, $\mathcal{TN}_2$ **after** TP $\mathbf{MUST}_{pos}$ **TP**.

By symmetry, we have that $TCT(\mathcal{TN}_1) \sim_{ct} TCT(\mathcal{TN}_2) \Rightarrow \mathcal{TN}_1 \sim_{pos} \mathcal{TN}_2$. □

Software simulation of the information web-system with regulation of access to Internet content*

Konstantin Budnikov and Alexander Kurochkin

Institute of Automation and Electrometry of Siberian Branch of the Russian Academy of Sciences, Academician Koptyug ave. 1, 630090 Novosibirsk, Russia
budnikov@iae.nsk.su, kurochkin@iae.nsk.su

Abstract. Filtering of requests to an Internet resource by its URL is the most popular way to restrict access to information hosted on the network, which is necessary to comply with the requirements of security, copyright, labor regime, etc. This method can be considered as the most balanced in terms of the advantages and disadvantages. It allows to use a selective approach to information resources located on one and the same IP-address and to prohibit access when necessary.

We have developed specialized software and have carried out simulation computer modeling of the information web-system, in which a filtering device is built in. Filter processes request flows to the server by a group of users connected to it and counter information from the server. In order to eliminate the influence of the network infrastructure on the operation of the model, the network interfaces were emulated by software, and all network data flows proceeded in the memory of the simulating computer. In the process of modeling, we compared 2 filtering methods: the standard method of preliminary analysis of requests and the proposed post-analysis method.

Computer simulation showed a decrease of the average waiting time for a response from the web-server when a user request to a web-resource passed through the emulated filtering device, which worked in the post-analysis mode, up to 14% compared to the device that worked in standard mode. Filter throughput increased by up to 54%.

Keywords: software simulation, web-system, HTTP filters.

1 Introduction

The growth of the Internet in conjunction with the advent of a large number of information Internet resources, access to which needs to be restricted for a number of reasons, including age and moral and ethical criteria, requirements for compliance with security, copyright, labor regime, etc., requires improvement of methods and tools of ensuring a selective ban on access to information on the network. Currently, these methods include: restricting access by IP address, by URL, changing requests to DNS

* Work is supported by IAE SB RAS, project #AAAA-A17-11706061006-6

servers, using proxy servers, packet filtering. These approaches combine both advantages and disadvantages.

Filtering of requests to a resource by its URL is the most popular method, which can be recognized as the most balanced in terms of its strengths and weaknesses. It allows for a selective analysis of requests to information resources located on one IP-address and to prohibit access to a resource when necessary. For a separate Internet access device (computer, smartphone, tablet), a specially installed program performs the filtering process [2; 3], and for a group of devices - a filtering device with Internet access to which they are connected [4-7]. In the framework of this work, we consider a model of an information web-system in which the filtering device processes information flows from a group of connected subscribers.

The standard filtering algorithm by the URL, which is described, for example, in patents Cisco Technology, Inc. [5] and Ironport Systems, Inc. [6], presupposes a preliminary check of the request at the input of the device, and only by its results the request either passes further or is blocked. The verification itself takes the time associated with intercepting the request, extracting the URL from it and searching for it in the lists of forbidden addresses. At this time, the request is delayed by the filter.

Working on the improvement of this algorithm, we came to an algorithm that was called a filtering algorithm with post-analysis of requests [7, 8]. We expected that it would provide a reduction of the waiting time for a response from the web-server and an increase of the throughput of the filtering device and as a result of the system as a whole. To confirm this assumption, we carried out a computer simulation, which will be discussed later.

2 Algorithms of filtration

The standard filtering algorithm mentioned above assumes the following sequence of actions.

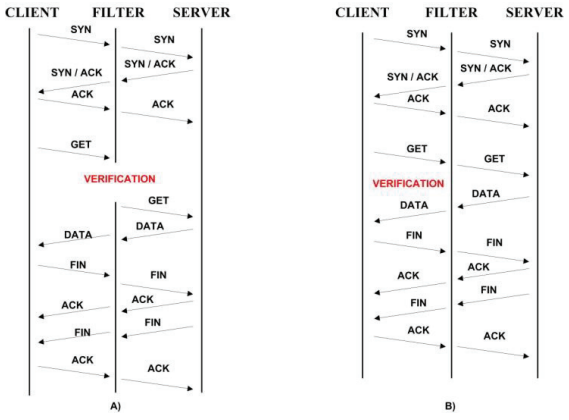


Fig. 1. Time diagrams for filtering of requests to the web-server using the standard algorithm (A) and the algorithm for post-analysis of requests (B).

A device that filters by an URL intercepts a user's request passing through it, extracts from it the address of the resource being accessed. Further, depending on the built-in algorithm, this address is searched in the lists of resources that are prohibited or allowed. If the URL that is being accessed is allowed, the request is passed to the Internet, reaches the server with the required resource, and the server returns a response with the requested information. If access to a resource of user interest is denied, the request is blocked by the filter. The timing diagram of events that occur during filtering by this algorithm is shown in Fig.1 (A).

The improvement of this algorithm using post-analysis of requests gives a gain in the time of a user request passing through a filtering device compared to the standard packet processing sequence. It consists of the time spent on determining the TCP session for each packet, forming a user request from the packets, extracting the identifier of the requested resource URL and checking the request for access to the requested resource using internal lists of forbidden URLs. The timing diagram of events that occur during filtering by this algorithm is shown on Fig.1 (B). Both algorithms are discussed in detail in [8].

3 Model of the system

A simulated web-system (Fig. 2) consists of a web-server, its clients, sending requests to web-resources located on it and a filtering device that regulates clients' access to resources. The task of its modeling is focused on the problem of the fastest passing requests through the filter. Model of the filter consists of two equivalent channels that provide packet traffic through the device and its filtering. Each channel contains the following modules (Fig. 2): network packet reading (MPR), packet sorting (MPS) and packet transfer (MPT). The central module of the model is a packet analyzer (MAP) common to both channels. Interaction with communication lines occurs through the modules of network interfaces MNI1 and MNI2. The operation of the filtering device in the simulated system is described in more detail in [9].

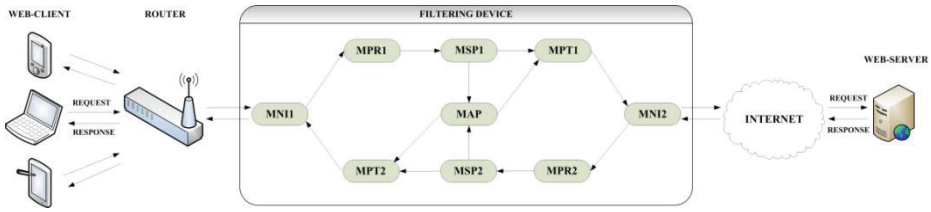


Fig. 2. Simulated web-system.

Since the main focus of the research is on the problem of the fastest passing requests through the filter, the simulation process compared 2 filtering algorithms mentioned above. In order to get the result of the comparison, which would not depend on the communication lines that connect the components of the system, but contained only a comparison of the processing times of requests to the web-server and its re-

sponses, it was decided to simulate the system, putting requests, responses and filtering device itself into a computer memory.

4 Simulation of the system

The simulation process takes place using preliminary prepared files containing network packets recorded in advance. They are exchanged between the web-client and the web-server during an HTTP session. In the above model, we replaced the network interface modules (MNI) with network simulation modules (MNS). Other modules remained unchanged. Both MNS have the same architecture and operation algorithm, and the role of each module as a web-client or server is set when the simulator of the system is configured. In the process of simulation, the MNS uses the above-mentioned file containing network packets that the web-client exchanged with the server during the HTTP session. Each packet in such a file has a number of attributes, the main one being the identifier of the sender of the packet. Both MNS perform the main work on simulation. Each of them consists of four main components (Fig. 3):

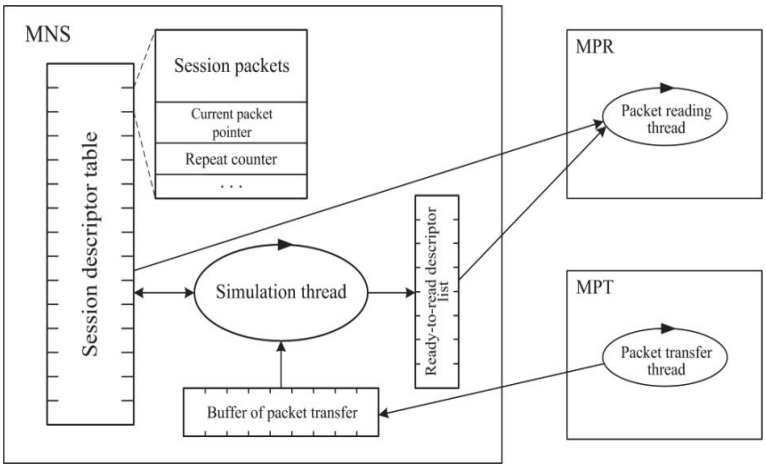


Fig. 3. The structure of the network simulation module and its interaction with MPR and MPT.

1. Simulation thread, which provides capability of the module to simulate the operations of receiving and transmitting packets.
2. Session descriptor table. This table contains all the information about the process of emulation of HTTP sessions during the test. Each descriptor contains an image of a file with session packets, a pointer to the current packet being processed, a repetition counter for this session, and a number of other auxiliary parameters. The size of the table, the number of repetitions and the file with packages are set in the parameters of the test.
3. Buffer of packet transfer accumulates packets devoted for transmission. Discipline of packet service is consistent with the principle of FIFO. Packages come from

MPT and are added to the end of the list. The internal simulation thread reads packets from the list, starting with the first one.

4. A list of descriptors of sessions which are ready for reading. This list contains descriptor numbers in which the current packet corresponds to the incoming packet for the current session state. Numbers are added by the simulation thread to the end of the list. When requesting a packet, the MPR selects a descriptor number randomly and reads the corresponding current packet.

The algorithm of the simulation thread work is shown in Fig.4.When a simulation experiment runs, both MNS are loaded into memory and initiated. As a part of this process, they get their roles: the client or server, their descriptor tables are being formed. Packages from the specified session file are recorded into each descriptor and modified so that, first, the session is unique with respect to other descriptors, and second, the task of determining the package to the descriptor is most simplified. The current packet pointer is set to the first packet. The repeat counter is reset. The packet transmission buffer is created empty. At startup, the simulation stream is set to the waiting state of the packet for transmission.

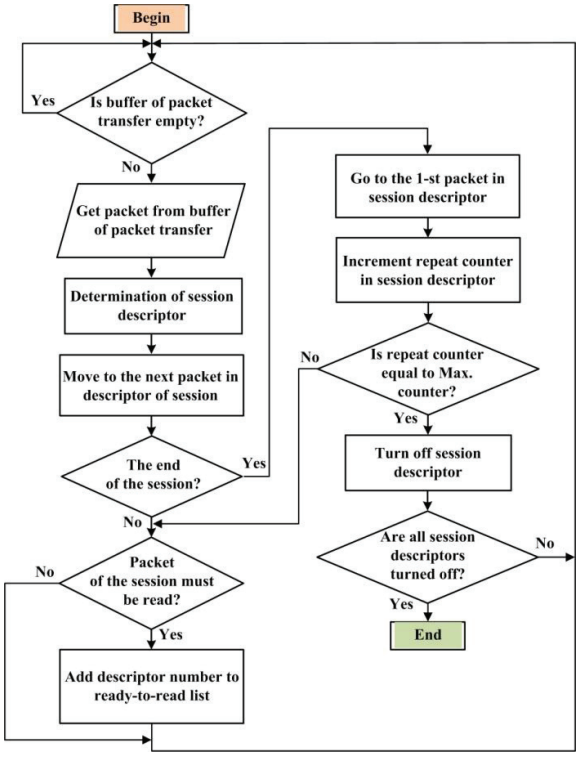


Fig. 4. Work flow chart for algorithm of the simulation thread.

The list of descriptors of session which are ready for each MNS is formed differently. In the MNS, which emulates the exchange with a web-server, the list is created empty, and when the MNS of web-client is loaded, the numbers of all descriptors are entered in this list. Therefore, after the launch of the corresponding read thread in the MPR, the processing of HTTP session packets begins. They are fed to the MPR input. The main cycle of the simulation process starts.

The simulation thread is activated when the MPT module adds a packet to the buffer of packet transfer. The packet is read from the buffer and determined to belong to one of the session descriptors. The simulation thread goes to the next packet in the session (increments the current packet pointer in the descriptor). The sender ID of this packet is analyzed. If the packet needs to be sent to the MPR, then the descriptor number is added to the list of ready-to-read descriptors. Next, the thread checks for data in the packet transfer buffer. If unprocessed packets are present, they are processed in the same way. If the buffer is empty, then the simulation thread pauses. In case of reaching the end of the session after the increment of the current packet pointer, this pointer is redirected to the first packet in the session and the repetition counter in the descriptor increases. If the repetition counter reaches the value specified in the test parameters, this session descriptor is disabled and no longer participates in the simulation process. The experiment is considered complete when all descriptors are disabled.

Reading of packets from the session descriptor table follows a similar algorithm. The packet reading thread loads the number of one of the descriptors that is ready for reading and copies the packet into the specified memory area. It then moves to the next packet in the session and analyzes the sender ID of this packet. If the packet also needs to be sent to the MPR, the number of the corresponding descriptor is returned to the list of ready-to-read descriptors. Packet that is read goes to the MPR, and the reading thread continues processing of descriptors. When the end of the session is reached, the current packet pointer is set to the first packet and the session repeat counter increases. The corresponding descriptor is no longer serviced when the repeat counter equals the value specified in the experiment parameters.

5 Simulation results

Using the above presented web-system simulator, we compared the filtering algorithms mentioned earlier. This is a standard filtering algorithm by URL and the improved algorithm based on post-analysis of requests to a web-resource. In the process of testing, the work of the system was simulated with different intensities of request streams from clients and sizes of responses from the web-server ranging from 1Kbytes to 100Kbytes. The emulation was done using an office computer with an Intel Core i7 870 4 × 2.93Ghz processor and 4Gb of memory. The following system characteristics were investigated, such as: the number of processed HTTP sessions per 1 second, the intensity of virtual network streams, and the client's waiting time for a response from the web-server. In the process of testing, the virtual exchange rate reached 7.7Gbit/s (8770 requests per second) with the size of server responses of

100Kbytes. The number of requests per second reached 135600 (1.9Gbit/s) with the size of server responses of 1Kbytes.

Figure 5 shows graphs of the response time from the web-server (μs) as a function of the intensity of virtual network flows in the emulated system (Mbit/s) for standard filtering of requests (A) and filtering with post-analysis (B) for the size of the response from web-server of 1Kbyte.

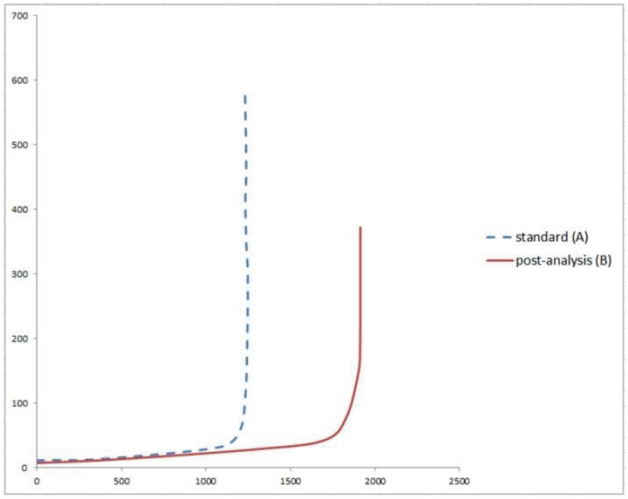


Fig. 5. Dependencies of response time from the web-server (μs) on the intensity of virtual network streams in the emulated system (Mbit/s). Size of the response from web-server equals to 1Kbyte.

Simulation showed a decrease in the average waiting time for a response from the web-server when a user request to a web-resource passed through an emulated filtering device, which worked in the post-analysis mode, compared to the device that worked in the preliminary query analysis mode. At the end of the initial segment of the graph (its linear part) for the standard (A) algorithm, it reached 14% for server responses of 1Kbyte and 11% for server responses of 100Kbytes. Filter throughput increased by up to 54% with 1K server responses and up to 22% with 100K server responses. The obtained values refer to a specific modeling computer and the model of the system that is implemented on it. In other cases, results can be obtained that differ from the shown above.

6 Conclusion and future work

Computer simulation allows you to find solutions for problems that cannot always be solved, or their solution is difficult using other modeling methods. In our case, this is a computer simulation of the information web-system, which has a built-in filtering device. The simulation was performed using specially created software that emulates

a filtering device, network interfaces, communication lines, clients, and a web-server. This allowed to exclude the network component of the system by placing a filtering device in the center of the model. Functionally, the filter model was not changed when the algorithm was changed. Thus, it was possible to compare filtering algorithms by conducting experiments on the same computer with the same set of processed data. In the process of modeling, we compared the standard filtering algorithm and the improved algorithm using the post-analysis of requests. The advantage of the improved algorithm and its perspective of use in filtering devices was confirmed. The relatively small gain in waiting time for a response from a web server can be interpreted as the need for hardware support for an improved filtering algorithm. Further development of the filter is seen in the perspective of using an improved algorithm in conjunction with its hardware support.

References

1. Apetyan,S Kovalev, A., Veybach, A.: Filtering content in Internet. Analysis of international practice. Foundation of Civil Society Development, 22 May, 2013. http://civilfund.ru/Filtraciya_Kontenta_V_Internete_Analiz_Mirovoy_Praktiki.pdf [in Russian]
2. Osipov, G.S., Tihomirov, I.A., Sochenkov, I.V.: Method and system for web content filtration. Patent RU 2446460 C1. IPC G06F 21/20 (2006.01), published 27.03.2012. Official Bulletin of Rospatent "Inventions. Useful Models", 2012, #9 [in Russian].
3. Bellinson, C., Evans, C., Fravert, H., Taylor, W.: "Content filtering for web browsing". Patent US20040006621 A1, IPC G06F17/30, G06F13/00, G06F15/00, G05B1/00, G06F17/00, published 08.01.2004.
4. Kumar, B., Sekhar, Ch., Kodukula, N.: PRE-EMPTIVE URL FILTERING TECHNIQUE. Patent N0.: US 8,838,741 B1 Int. Cl. G06F 15/16, Date of Patent: Sep. 16, 2014
5. Balasubrahmaniyan, J., Daftary, K., Yarlagadda, V. R., Kumar, K.: SYSTEM AND METHOD FOR URL FILTERING IN A FIREWALL. Patent US 20060064469A1, Int. Cl.G06F 15/16 (2006.01), Pub. Date: Mar. 23, (2006).
6. Bloch, E., Mohan, S., Pagaku, R.R., et al.: Apparatus for monitoring network traffic. Patent US 7849502 B1, Int Cl G06F 15/16 (2006.01), G06F 11/00 (2006.01), Pub. Date: Dec. 7, (2010).
7. Budnikov, K.I., Kuruchkin A.V.: Method of HTTP-packet flow filtration based on post-analysis of requests to Internet resource and filtering device for its realization. Patent RU 2 599 949 C1 IPC G06F15/00, G06F15/16, G06F 21/00, G06F 21/50. Official Bulletin of Rospatent "Inventions. Useful Models",,# 29, Published Oct. 10, (2016). [in Russian].
8. Budnikov, K.I., Kuruchkin, A.V., Lubkov, A.A., Yakovlev, A.V.: Regulation of access to web-resource based on post-analysis of http-requests. In: Kazanskiy, N.L. , Kudryashov, D.V. et al (eds.) Proceedings of the International conference Information Technology and Nanotechnology (ITNT 2016), Samara, Russia, May 17-19, 2016. CEUR Workshop Proceedings vol. 1638. Published on CEUR-WS: 02-Aug-2016. http://ceur-ws.org/Vol-1638/.
9. Budnikov, K.I., Kuruchkin, A.V., Lubkov, A.A., Yakovlev, A.V.: Experimental Study of Symmetric Computer Model of Http-Filter. In: Proc. of the 2018 3rd Russian-Pacific Conference on Computer Technology and Applications (RPC), 18-25 Aug. 2018, Vladivostok, Russia. doi: 10.1109/RPC.2018.8482147 https://ieeexplore.ieee.org/document/8482147.

Inter-country competition and collaboration in the miRNA science field *

Artemiy Firsov[1,3][0000−0002−7681−1032] and Igor Titov[2,3]

[1] Institute of Informatics Systems, 6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia https://www.iis.nsk.su
[2] Institute of Cytology and Genetics, Prospekt Lavrentyeva 10, Novosibirsk 630090, Russia http://www.bionet.nsc.ru
[3] Novosibirsk State University, 1, Pirogova str., Novosibirsk, 630090, Russia https://nsu.ru/

Abstract. Many digital libraries, such as PubMed, Scopus, appeared with the growth of the Internet: thus, many scientific articles became available in the digital form. We got an opportunity to query articles metadata, gather statistics, build co-authorship graphs, etc. This includes estimating the authors/institutions activity, revealing their interactions and other properties.
In this work we present the analysis of the characteristics of institutions interactions in the miRNA science field using the data from PubMed digital library. To tackle the problem of the institution name writing variability, we proposed the k-mer/n-gram boolean feature vector sorting algorithm -KOFER. We identified the leaders of the field - China, USA -, characterized the interactions and described the country level features of co-authorship. We observed that the USA were leading in the publication activity until China took the lead 4 years ago. However, the USA are the main co-authorship driver in this field.

Keywords: k-mer · n-gram · dbscan · identification · mirna · timsort · kofer · digital library · co-authorship.

1 Introduction

Many digital libraries appeared with the growth of the Internet, thus, the format of representation of many scientific articles changed. We got an opportunity to query articles metadata, gather some statistics, etc. This includes understanding the authors/institutions activity, their interactions, and other characteristics. One can also prove that the Paretto rule for the institutions publication activity holds true [2], or that the idea spreads from one author to another like the virus spreads from one person to the others. Having this information, we can further use it to predict the new science field creation, popularity of the particular science field. In general, it can be used in social informatics.

* The work of I.T. was supported by the Federal Agency of Scientific Organizations (project 0324-2019-0040).

Moreover, right now the new science field is emerging science of science [12]. It is a transdisciplinary field of science that aims to understand the evolution of ideas, choice of a research problem of particular scientist, etc. Without analyzing interactions between authors, institutions, and other, such a field just cannot exist.

However, to do that one should know to which real author/institution the authors name/affiliation from the paper corresponds to. The more precise correspondence we have, the better accuracy of statistics we can get. This disambiguation issue is not that simple considering big datasets, such as PubMed with $2 * 10^7$ articles. It becomes more complicated when you consider errors in the author name/affiliation made either by author, or by editor. Moreover, sometimes author/institution name might be changed, or the affiliation from the papers metadata may have mixed institution names for different authors. E.g. if the Author1 has Institute of Cytology and Genetics, Novosibirsk, Russia institution and the Author2 has Institute of Mathematics, Novosibirsk, Russia institution, their resulting affiliation for collaborative paper might be Institute of Cytology and Genetics, Institute of Mathematics, Novosibirsk, Russia. Moreover, affiliation can contain email, postal address and other insertions not related to institution name.

1.1 Author Disambiguation Overview

To disambiguate the authors, many sophisticated algorithms were proposed. Some base on the similarity function [13], others use clustering techniques [14], web information [15], etc. Different approaches are reviewed well in Ferreiras et al. paper [16].

Although they utilize different algorithms and methods, almost all of them have one thing in common they use affiliation as the author feature. That means that similarity between two different records of authors from different articles is measured using affiliation also. Moreover, there is no uniform algorithm to process affiliation entry, in different papers researchers use different similarity measures. However, one may think of another use of affiliation.

1.2 Affiliation Disambiguation Problem

We know that the researchers can work in different places, thus, can have multiple affiliations in their papers. We also know that they leave/get fired from/change their institution rarely. By that means, one may identify the author having the knowledge of his affiliations, or at least propose a hypothesis. On the other hand, affiliation is represented by its authors/workers. That means that having a set of author names, one can deduce, in which institution these authors work, or at least propose a hypothesis. I.e. people are the feature of institution and institutions are the feature of a person. Using this statement, we can think of the author disambiguation issue and the affiliation disambiguation issue as of two separate issues. Moreover, results of one issue solution can be used to enhance the results of the other.

Having this in mind, we in Molecular Genetics Department of Institute of Cytology and Genetics propose an idea that iterative, self-consistent solution of the author disambiguation issue, the affiliation disambiguation, and the paper topic extraction issue can increase the accuracy of all these issues.

In this paper, we aim to provide the solution for Affiliation Disambiguation problem. In addition, the whole self-consistent project is currently under development in the laboratory of molecular genetic systems in the Institute of Cytology and Genetics under I.I. Titov supervision.

The organization name disambiguation problem has already been addressed in several works, in which authors aimed to disambiguate organization names mined in the social web-data. Authors used web data [3], and sophisticated algorithms [4]. However, we are aiming to find the simple and yet precise solution, basing on the simple input data - just affiliations. That way we can get computationally effective algorithm, which can be specified using results of the author name disambiguation problem solution.

2 Methods and materials

2.1 Prerequisites

The basic idea of the work is to get the groups of organizations mentions, which contain only mentions of one institution. After that we may use that information to build the co-authorship graph of organizations/countries, get static and dynamic characteristics of the science field, etc. So on the first step we solved the clustering problem of institutions names writings:

$$maxf(C)\ subject\ to\ C = (C_1, ...C_k), C_1 \cup ... \cup C_k = S \tag{1}$$

,where S - the set of affiliations extracted from the publications, C_i - group of similar affiliations. So we want to assign a label to each of the affiliations in the dataset, so that the final grouping by labels maximizes some function f. This f can be constructed in many ways, however in our case, the closer the grouping is to the ground-truth grouping (i.e. one group contains all and only affiliations that refer to the same institution), the higher the value of the function. So the previous problem statement will transform:

$$maxf(C, R)\ subject\ to\ R = (R_1, ...R_k), C = (C_1, ...C_k) \tag{2}$$

$$C_1 \cup ... \cup C_k = S, R_1 \cup ... \cup R_k = S$$

, where R is the ground-truth grouping of the affiliation set S, labeled by the author, m the ground-truth number of labels. The function f that provides such characteristics is discussed in the Evaluation Metrics section of this chapter.

2.2 Dataset

To conduct experiments, we have gathered two datasets from PubMed [1] digital library using MEDLINE file format. First one is the Novosibirsk dataset,

that consists of the preprocessed affiliations of the Novosibirsk institutions. This dataset was gathered in the Titovs and Blinovs work [2] dedicated to the author disambiguation problem. We labeled this dataset to have the ground-truth affiliation clustering to further use it for clustering algorithm hyperparameters fine-tuning. Second one is the miRNA dataset gathered on the following search query on the PubMed website over *Title* and *Abstract* fields:

(((((((miRNA) OR mi-RNA) OR microRNA) OR micro-RNA) OR miRNAs) OR mi-RNAs) OR microRNAs) OR micro-RNAs.

The miRNA dataset contains the publications available on the PubMed digital library as of 11/11/2018.

Table 1: Characteristics of the datasets used in the work

	Novosibirsk dataset	miRNA dataset
# of articles	$\sim$ 6,000	77,800
Year	... - 2014	... - 2018
# of affiliations	951	387,793
# of unique organizations	62	$\sim$ 20,000

2.3 Evaluation metrics

We used homogeneity (h), completeness (c) and v-measure score (v) [5] to evaluate the clustering results for Novosibirsk dataset.

$$h = 1 - \frac{H(C|K)}{H(C)} \tag{3}$$

$$c = 1 - \frac{H(K|C)}{H(K)} \tag{4}$$

$$v = 2 * \frac{h * c}{h + c}, \tag{5}$$

where $H(C|K) = -\sum_{n=1}^{|C|}\sum_{n=1}^{|K|}\frac{n_{c,k}}{n} * log(\frac{n_{c,k}}{n_k})$, $H(C) = -\sum_{c=1}^{|C|}\frac{n_c}{n} * log(\frac{n_c}{n})$; $H(K|C), H(K)$ are constructed the same way.

These metrics are analogous to precision, recall and f-metric used in supervised learning. Homogeneity equals one if every cluster contains only all data points from one class. Completeness equals one if all data points from one class are assigned to the same cluster for every cluster. In addition, v-measure is derived from homogeneity and completeness. The closer these metrics are to one, the better the solution is.

Although we aimed to increase this parameters, it is pretty hard to have them be near 1 if you work with affiliation entry only. So, we introduced additional metric to choose from different clustering results. As we aim to create an

instrument that reveals some statistics of institutions activity basing on their names only, it is reasonable to try to cluster affiliations in the way that final cluster count is equal to ground-truth class count. Thus, we chose those hyper-parameters that gave clusters count close to real class count. All the clustering quality metrics were calculated using the most significant clusters (cluster volume ≥ 10), as we want to be aware only on those institutions that are actively publishing in a certain field.

2.4 Data preprocessing

Pre-processing stage of all algorithms is performed before actual calculations. As discussed in introduction, affiliations have some additional information, which relate to the author, not institution. This leads to the big number of variations of institution name writing. On that stage, we remove emails/zip/phone/numbers from affiliation using regular expressions. We also perform standard operations, like preserving only alphabetical characters, expanding the abbreviations, removing stopwords, etc.

During the research we tried several NLP frameworks hoping they can fix errors described in some points above. These include NLTK [7], language_check [8] and others. We found out that affiliations (or institutions references) cannot be fully considered as manifestations of natural language, and NLP frameworks perform poorly on them, giving a lot of errors on each affiliation. However, they still can be used to fix errors inside words such as institue and others.

So far we eliminated explicit artifacts, however, there may be implicit artifacts, like name of the laboratory where the author works. We call this an implicit artifact as this name can be different for different institutions and it is hard to provide deterministic algorithm that will work for every situation.

To handle such artifacts, we introduced regular expression based algorithm based on keywords and country names. We provide our algorithm with keywords, such as center, institute, etc. We also provided our algorithm with full list of country names. Now, having all that information, we can represent our affiliation as a sequence of numbers. We split affiliation by commas and assign each part 0 number. If any keyword is in a particular part, we assign it number 1. If any country is in a particular part, we assign it number 3. Thus, Institute of Cytology and Genetics, Novosibirsk, Russia is represented as 103; Institute of Cytology and Genetics, Institute of Mathematics, Novosibirsk, Russia is represented as 1103. Number of ones represents the number of institution names in the affiliation, which can further be handled with a simple regular expression (0?1+0?0*0?3?0?).

The advantage of such approach is that new notions can be introduced into this algorithm, e.g. cities. Moreover, it is extensible and modifiable, as providing new keywords and country/city names, we can wider our algorithm configuration to work with bigger domain of affiliations.

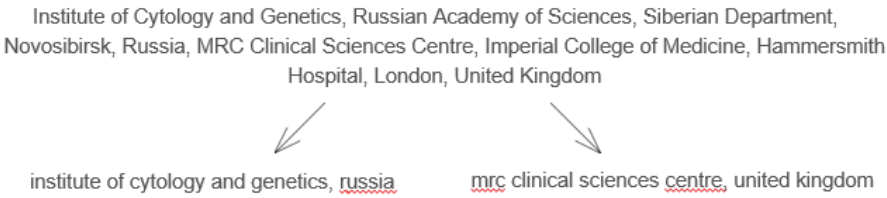


Fig. 1: Example of normalized and split affiliation

2.5 Clustering and similarity

After the pre-processing stage, the clustering stage is performed. We tried different techniques for the clustering K-Means and DBSCAN [10] and different popular similarity functions Levenshtein, Jaccard, Smith-Waterman. We used scikit-learn [9] implementation of those.

We also tried using K-Mers feature vectors to find similarity between affiliations. K-Mer is a notion that came from genetics. It is a substring of a certain string of length K. Geneticists use it to analyze DNA/RNA sequences. In Natural Language Processing there is a similar notion n-gram. Building K-Mer feature vector is described below and in section K-Mers Boolean Feature Vector Sorting (KOFER) based Clustering. However, it is important to notice that we used letter K-Mers, not word K-Mers in the work.

Similarity functions were used to create the distance matrix, and also K-Mers were used to create features of a certain affiliation. As K-Mer is a substring of a string of length K, one can assign the Boolean vector to the affiliation. In this vector each bit represents the presence of a certain K-Mer in the affiliation. As K-Mer dictionary power exponentially grows with the K number, this dictionary upon K-Mers consists only of K-Mers present in the certain dataset that is being processed in the experiment.

Basically, similarity function for strings is a function that takes as an input two terms and outputs the value between 0 and 1:

$$f(x, y) = z \tag{6}$$

, where $z \in [0, 1], x, y \in V^*, V$ - alphabet. However, for Boolean vectors - $x, y \in B^m$, m - the size of the K-Mer dictionary.

During experiments we found out that KMeans and DBSCAN perform poorly on affiliations data, so we proposed another method based on the K-Mer boolean feature vectors clustering.

The idea of the method is based on the consistency of affiliation writing. E.g. if an author works in "Institute of Cytology and Genetics", it is highly likely that this particular words with some additional information will be present as affiliation in his work. Moreover, we can assume that these words should usually be placed in the "first position" of affiliation, like in "Novosibirsk Institute of Cytology and Genetics, Novosibirsk, Russia".

The naive idea would be to sort affiliations strings, find distance between current and next neighbors, and then set a threshold for the distance. If the distance exceeds the threshold, we can consider this pair to belong to different clusters, e.g. table 2.

Table 2: 10 affiliation entries sorted by name. Left column - affiliation. Right column - demonstrative distance. 5th line has high distance as affiliation refer to different organizations. The last row is automatically assigned to the latest cluster as there is no next row to calculate similarity with.

institute of cytology and genetics	0.1
institute of cytologyand genetics	0.2
institute of cytology and gnetics	0.3
instiute of cytologyand genetics	0.2
institute of cytology and genetics	**0.8**
institue of bioorganic chemistry	0.1
institue of biorganic chemistry	0.2
institue of bioorganicchemistry	0.1
institue of bioorganic chemistry	0
institue of bioorganic chemistry	

Such an approach benefits in performance time. DBSCAN complexity is $O(n^2)$ in the worst case (running ahead, we note that we observed such situation running experiments), as well as sorting (however we did not observe that problem during experiments). Distance calculation time grows linearly with increasing number of records, as well as comparison time. Thus, performance complexity would be $O(nlog(n))$ ideally. However, the nave approach performs badly in the following situations:

1. There exists preceding part in affiliation name Institute of Cytology and Genetics vs Novosibirsk Institute of Cytology and Genetics.
2. There is an error in the beginning of the affiliation Lnstitute of Cytology and Genetics.

In both cases, entries are assigned into different clusters, as they start with different characters.

2.6 K-Mer Boolean vector sorting

To tackle the problems pointed above, we decided to perform sorting on K-Mers vector instead of plain text. Here and further, we mean letter K-Mer when mention K-Mer, not word K-Mer. K-Mer vector can be constructed in different ways, but firstly, one should calculate K-Mer dictionary:

1. Take the dataset with affiliations
2. Calculate K-Mers for each string

3. Take only unique K-Mers and reorder them from frequent to rare

We need reordering to provide each K-Mer with its place and to provide position invariance.

Having this done, we then proceed to feature vector calculation, which can be done using one of the following approaches:

1. Create the binary vector, that represents the presence of all K-Mers in the affiliation
2. Create natural vector, that represents the number of occurrences of all K-Mers in the affiliation

Further, we discuss the first approach and provide the results of clustering using binary K-Mer features. Then, we can lexicographically sort these vectors so that affiliations with similar contents will be aligned together in the array of all affiliations. And moreover, this procedure can restore the conformity between affiliation substrings, as we can see from the 4.

E.g., assume that the dataset contains only two words "institute" and "institue". We use the simple example here for the ease of understanding. Then the K-Mer dictionary and K-Mer Boolean feature vectors would look like this:

Table 3: K-Mers dictionary and K-Mers boolean feature vectors for the simple case

Dictionary	**K-Mer**	**in**	**ns**	**st**	**ti**	**it**	**tu**	**ut**	**tu**	**ue**
	Occurrences	2	2	2	2	2	2	1	1	1
For the word "institute"		1	1	1	1	1	1	1	1	0
For the word "institue"		1	1	1	1	1	1	0	0	1

Having this done, we can further lexicographically sort the K-Mer Boolean feature vectors and find the distance between two neighboring vectors using Boolean distance metrics, as we did in the Results chapter, for the use in clustering by the threshold. For example, below is the table showing words sorted by their K-Mer Boolean feature vectors representation. Here, we use five words to show how we reach the threshold of the distance in the sorted list of vectors - "institute", "institute", "institute", "center", "centre". We also eliminate the explanation of the K-Mers dictionary construction, as it was explained before. The distance was calculated using Dice distance [11].

We can see that similar words grouped and the distance reaches its peak when there is a change from the word "institue" to the word "center". If we than say that the distance threshold should be bigger than 0.43 to consider previous and further records to belong to different clusters, we than can validly assign different words to different clusters.

Table 4: Affiliations sorted lexicographically by the K-Mer booleam feature vectors with distances between neighboring records calculated with Dice distance

Word	K-Mer Boolean feature vector	Distance between current and next
institute	1111111100000000	0.2
insitute	1111100100001000	0.43
institue	1111011000010000	**0.83**
center	0000100011100100	0.4
centre	0000000011100011	0.2

2.7 Country Identification

To identify countries in affiliations we used the open data [17] with the list of countries and cities provided. If the country or city was present in the affiliation, than the affiliation was assigned the corresponding country

3 Results

In this paper, we present the results for country level co-authorship in the miRNA field. Using the K-Mer Boolean vector sorting algorithm we were able to cluster the miRNA affiliations data. From the 387,793 affiliations we got 23,655 clusters. i.e. institutions.

3.1 PubMed statistics

To have the properties to compare with, we got the statistics from the PubMed website in Fig.2. All the plots were generated using the matplotlib software [19]. The growth of the articles available in the PubMed is different from the ones foe the miRNA science field - for all the publications in the miRNA field the linear model does not fit. The beginning of the growth is different from the remainder part.

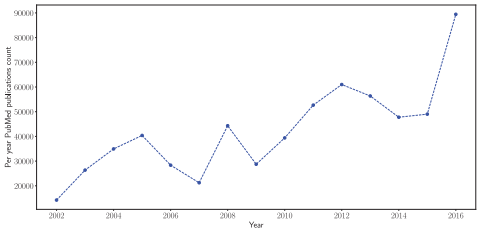
The logistic function estimation on the remainder part gives

$$\frac{a}{1 + b * \exp^{-c*x}} = \frac{19603.09}{1 + 31.09 * \exp^{-0.42*x}} \tag{7}$$

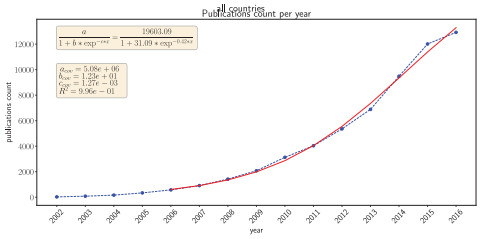
parameter values. And the exponential estimation at the beginning

$$\exp^{a*x+b} = \exp^{3.62*x+0.74} \tag{8}$$

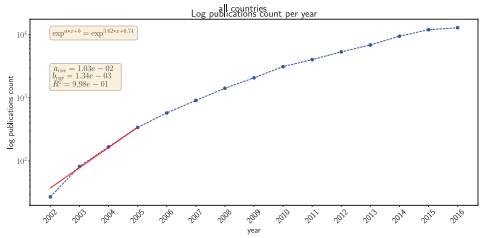
This shows that likely the miRNA field is in the saturation state. All the graphs are built using the 2003 - 2016 data, because the 2017, 2018 years are the years when the field reaches it's plateau.



(a) Number of publications added to the PubMed digital library per year



(b) Number of publications per year



(c) Log number of publications per year

Fig. 2: Publications in the miRNA science field

3.2 Countries publication activity

It would be interesting now to see per country publication activity (Fig.3). We may see that the USA had the rapid start of publication in this field, reaching 100 publications in 2004. However, the growth started to reduce over time, whereas China had higher growth, which led to China becoming the new leader in 2013. The numeric growth estimations for separate countries are available in Fig.4 and Tab.5.

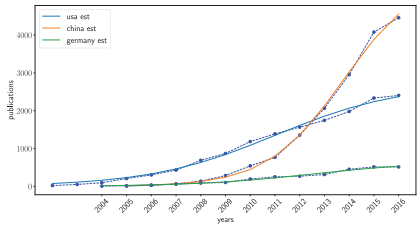


Fig. 3: Comparison of the countries publication activity

The logistic parameters estimations for publication activity of different countries is presented in the table below:

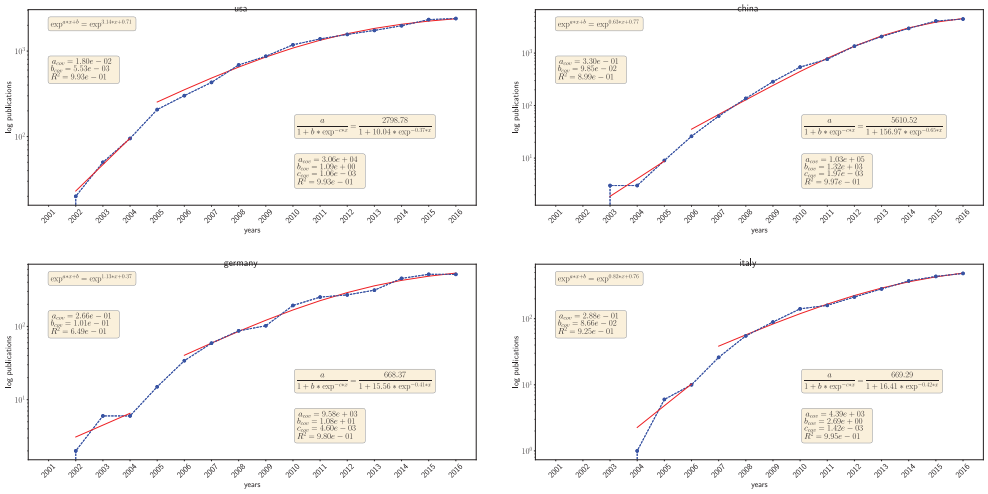


Fig. 4: Countries publication activity with numeric estimations.

Table 5: Parameter of the logistic function estimation for publication activity of different countries

Country	a	b	c
Australia	397,04	44,51	0.49
Canada	396,36	36,44	0.42
China	5,610.52	156,97	0.65
France	270,78	56,34	0.42
Germany	668,37	15,56	0,41
India	300,92	93,69	0.61
Italy	669,29	16,41	0,42
Japan	400,22	25,3.1	0.54
Korea	277,04	150,24	0,68
Netherlands	237,04	10,57	0.38
Spain	268,22	47,5	0.47
Switzerland	193,1	14.26	0.38
UK	961,07	29.11	0.36
USA	2798,78	10,04	0.37

3.3 Countries interaction graph

To see how countries interact, we have built the graph using the gephi software [18]. We used only those countries that have published more than 500 articles to reduce the noisiness of the graph in Fig. 5.

The graph shows that the USA and China are the leaders in this field, as well as they publish together a lot. Although connections are quite dense, and there are many joint publications between different country pairs, the number of joint publications seems to be quite low, and it is not clear whether some country is the driver of joint publications, or it is the common practice in the field to publish together. This problem will be addressed in the 3.4

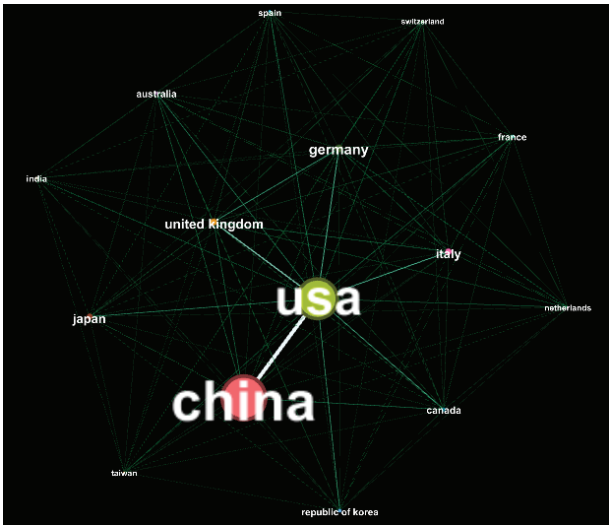


Fig. 5: Co-authorship of countries in the miRNA field. The label text, as well as the size of the point reflects the number of published articles, edges thickness shows the number of articles published together. If there were more than 2 countries in the publication, each pair of countries is considered to have had the joint publication.

3.4 Joint publications

During the 2002-2016 period, there were 52,407 publications, 5,412 of which were international, i.e. joint. The field until some time did not have much joint publications, however things changed in 2013. That year USA and China started actively publish together (Fig. 6).

However, the main driver for joint publications is the USA, as it has more publications with different countries than China (Tab.6). Also, it is interesting

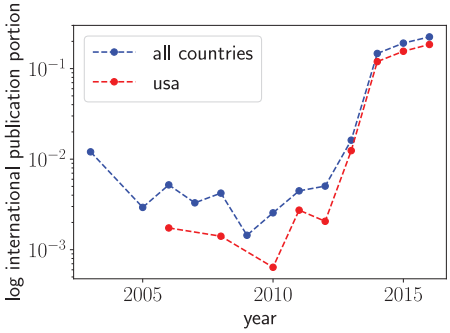


Fig. 6: The log portion of international publications relative to overall number of publications

to notice that major part of all USA joint publications appeared after the 2013, when it started actively publishing with China.

Table 6: Most active countries pairs sorted by the number of joint publications

Country 1	**Country 2**	**Publications**
USA	China	1,084
USA	UK	324
USA	Italy	227
USA	Germany	223
USA	Canada	190
UK	Germany	182
USA	Korea	165
USA	Japan	145
USA	Australia	131
China	Canada	110
UK	Italy	103
China	UK	101
USA	France	100

4 Discussion

Getting the implicit properties of science field has major research interest as it reveals the current state of the field, provides the opportunity to compare different science fields with uniform instrument and gives the possibility to predict the creation of new fields or the future of the particular one.

And although metrics are the subject of the disputes - whether they are needed or not, useful or harmful - they are still of peoples interest. Whether

government or companies support the research, they also rely on information environment surrounding the science field, which is often quantified to, e.g. number of publications within a science field, the impact of the research on the market, etc. Having more possibilities to reveal the "true" state of the science field would help researchers to show the importance of their field, as well as the funding organizations to distribute their funds efficiently. Thus, this work aims to reveal the quantitative metrics of the science field.

In our worked we used K-Mer Boolean feature vector sorting algorithm, which performed fast, however it still has the drawback of splitting the cluster into 2 separate cluster if there appears the different affiliation with common k-mer/n-gram inside the cluster. This problem can be tackled using the numeric feature vector, which will consider not only the presence of the k-mer/n-gram, but also the count of them present in the affiliation.

This work yet does not cover the affiliation level properties of the miRNA science field. It would be interesting to see the leaders of the field, track their history and also get the properties of the co-authorship graph.

The points mentioned above will be considered in the upcoming paper.

5 Conclusion

In this work we have implemented the algorithm for fast institution name clustering based on the K-Mer Boolean feature vector sorting - KOFER. Using that algorithm we managed to cluster the miRNA science field affiliations data.

Using the clustering results, we were able to get properties of country level interactions, see that China is currently the leading country in this field, however the USA is the biggest driver of joint publications.

The linear growth model does not fit the publication activity of countries - the relaxation should be taken into account. That tells us that the field is currently reaching it's peak.

References

1. Ncbi.nlm.nih.gov. (2019). Home - PubMed - NCBI. [online] Available at: https://www.ncbi.nlm.nih.gov/pubmed/ [Accessed 17 Jan. 2019].
2. Titov, I., Blinov, A.: Research of the structure and evolution of scientific co-authorship based on the analysis of Novosibirsk institutes publications in the biology and medicine science field. Vavilov Journal of Genetics and Selection, 2014 vol. 18 4/2
3. Shu Zhang, Jianwei Wu, Dequan Zheng, Yao Meng, Hao Yu: An Adaptive Method for Organization Name Disambiguation with Feature Reinforcing. 26th Pacific Asia Conference on Language, Information and Computation, 2012, p.237-245
4. Nafiye Polat: Experiments on company name disambiguation with supervised classification techniques. 2013 International Conference on Electronics, Computer and Computation (ICECCO), 2013. doi:10.1109/ICECCO.2013.6718248
5. Bell Hirschberg, J. and Rosenberg, A. (2007). V-Measure: A conditional entropy-based external cluster evaluation. In: EMNLP. Prague.

6. Rajaraman A, Ulman J (2011) Data Mining. In: I.stanford.edu. http://i.stanford.edu/ ullman/mmds/ch1.pdf. Accessed 20 Jan 2019
7. (2018) Natural Language Toolkit NLTK 3.4 documentation. In: Nltk.org. https://www.nltk.org/. Accessed 20 Jan 2019
8. (2017) language-check. In: PyPI. https://pypi.org/project/language-check/. Accessed 20 Jan 2019
9. (2019) scikit-learn: machine learning in Python scikit-learn 0.20.2 documentation. In: Scikit-learn.org. https://scikit-learn.org/stable/. Accessed 20 Jan 2019
10. Ester M, Kriegel H, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. AAAI Press
11. Dice, Lee R (1945) Measures of the Amount of Ecologic Association Between Species
12. Fortunato S, Bergstrom C, Brner K, Evans J, Helbing D, Milojevi S, Petersen A, Radicchi F, Sinatra R, Uzzi B, Vespignani A, Waltman L, Wang D, Barabsi A (2018) Science of science. Science. doi: 10.1126/science.aao0185
13. Cohen W W, Ravikumar P D, Fienberg S E (2003) A comparison of string distance metrics for name-matching tasks. IIWeb
14. Kamber, Han J. (2005) Data mining: concepts and technique. Morgan Kauffman
15. Jain A K, Murty M N, Flynn P J (1999) Data clustering: a review. ACM Computing Surveys, vol. 31
16. Ferreira A A, Gonalves M A, Laender A H F (2012) A Brief Survey of Automatic Methods for Author Name. SIGMOD Record, vol. 41
17. Usoltcev E (2016) meMo-Minsk - Overview. In: GitHub. https://github.com/meMo-Minsk. Accessed 25 Jan 2019
18. (2019) Gephi - The Open Graph Viz Platform. In: Gephi.org. https://gephi.org/. Accessed 25 Jan 2019
19. (2019) Matplotlib: Python plotting Matplotlib 3.0.2 documentation. In: Matplotlib.org. https://matplotlib.org/. Accessed 25 Jan 2019

Distributed Representation of n-gram Statistics for Boosting Self-Organizing Maps with Hyperdimensional Computing*

Denis Kleyko[1], Evgeny Osipov[1], Daswin De Silva[2], Urban Wiklund[3], Valeriy Vyatkin[1], and Damminda Alahakoon[2]

[1] Luleå University of Technology, Luleå, Sweden
{denis.kleyko, evgeny.osipov, valeriy.vyatkin}@ltu.se
[2] La Trobe University, Melbourne, Australia
{d.desilva, d.alahakoon}@latrobe.edu.au
[3] Umeå University, Umeå, Sweden
urban.wiklund@umu.se

Abstract. This paper presents an approach for substantial reduction of the training and operating phases of Self-Organizing Maps in tasks of 2-D projection of multi-dimensional symbolic data for natural language processing such as language classification, topic extraction, and ontology development. The conventional approach for this type of problem is to use n-gram statistics as a fixed size representation for input of Self-Organizing Maps. The performance bottleneck with n-gram statistics is that the size of representation and as a result the computation time of Self-Organizing Maps grows exponentially with the size of n-grams. The presented approach is based on distributed representations of structured data using principles of hyperdimensional computing. The experiments performed on the European languages recognition task demonstrate that Self-Organizing Maps trained with distributed representations require less computations than the conventional n-gram statistics while well preserving the overall performance of Self-Organizing Maps.

Keywords: Self-Organizing Maps, n-gram statistics, hyperdimensional computing, symbol strings

1 Introduction

The Self-Organizing Map (SOM) algorithm [23, 37] has been proven to be an effective technique for unsupervised machine learning and dimension reduction of multi-dimensional data. A broad range of applications ranging from its conventional use in 2-D visualization of multi-dimensional data to more recent developments such as analysis of energy consumption patterns in urban environments [6, 8], autonomous video surveillance [27], multimodal data fusion [14],

* This work was supported by the Swedish Research Council (VR, grant 2015-04677) and the Swedish Foundation for International Cooperation in Research and Higher Education (grant IB2018-7482) for its Initiation Grant for Internationalisation, which allowed conducting the study.

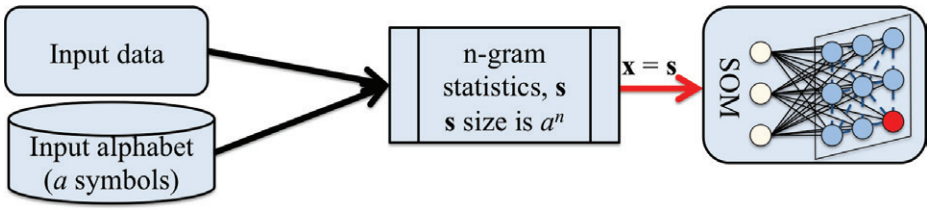


Fig. 1. Outline of the conventional approach.

incremental change detection [26], learning models from spiking neurons [12], and identification of social media trends [3, 7]. The latter use-case is an example of an entire application domain of SOMs for learning on symbolic data. This type of data is typically present in various tasks of natural language processing.

As the SOM uses weight vectors of fixed dimensionality, this dimensionality must be equal to the dimensionality of the input data. A conventional approach for feeding variable length symbolic data into the SOM is to obtain a fixed length representation through n-gram statistics (e.g., bigrams when $n = 2$ or trigrams when $n = 3$). The n-gram statistics, which is a vector of all possible combinations of n symbols of the data alphabet, is calculated during a pre-processing routine, which populates the vector with occurrences of each n-gram in the symbolic data. An obvious computational bottleneck of such approach is due to the length of n-gram statistics, which grows exponentially with n. Since the vector is typically sparse some memory optimization is possible on the data input side. For example, only the indices of non-zero positions can be presented to the SOM. This, however, does not help with the distance calculation, which is the major operation of the SOM. Since weight vectors are dense, for computing the distances the input vectors must be unrolled to their original dimensionality. In this paper, we present an approach where the SOM uses mappings of n-gram statistics instead of the conventional n-gram statistics. Mappings are vectors of fixed arbitrary dimensionality, where the dimensionality can be substantially lower than the number of all possible n-grams.

Outline of the proposed approach

The core of the proposed approach is in the use of hyperdimensional computing and distributed data representation. Hyperdimensional computing is a bio-inspired computational paradigm in which all computations are done with randomly generated vectors of high dimensionality. Fig. 1 outlines the conventional approach of using n-gram statistics with SOMs. First, for the input symbolic data we calculate n-gram statistics. The size of the vector $\mathbf{s}$, which contains the n-gram statistics, will be determined by the size of the data's alphabet a and the chosen n. Next, the conventional approach will be to use $\mathbf{s}$ as an input $\mathbf{x}$ to either train or test the SOM (the red vertical line in Fig. 1). The approach proposed in this paper modifies the conventional approach by introducing an additional

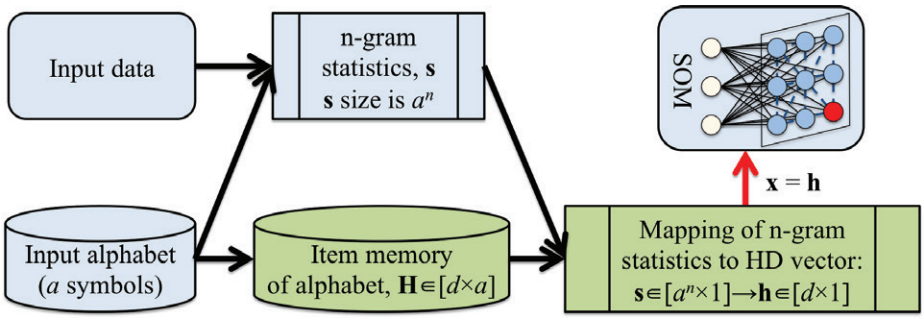


Fig. 2. Outline of the proposed approach.

step, as outlined in Fig. 2. The blocks in green denote the elements of the introduced additional step. For example, the item memory stores the distributed representations of the alphabet. In the proposed approach, before providing **s** to the SOM, **s** is mapped to a distributed representation **h**, which is then used as an input to the SOM (the red vertical line in Fig. 2).

The paper is structured as follows. Section 2 describes the related work. Section 3 presents the methods used in this paper. Section 4 reports the results of the experiments. The conclusions follow in Section 5.

2 Related Work

The SOM algorithm [23] was originally designed for metric vector spaces. It develops a non-linear mapping of a high-dimensional input space to a two-dimensional map of nodes using competitive, unsupervised learning. The output of the algorithm, the SOM represents an ordered topology of complex entities [24], which is then used for visualization, clustering, classification, profiling, or prediction. Multiple variants of the SOM algorithm that overcome structural, functional and application-focused limitations have been proposed. Among the key developments are the Generative Topographic Mapping based on non-linear latent variable modeling [4], the Growing SOM (GSOM) that addresses the predetermined size constraints [1], the TASOM based on adaptive learning rates and neighborhood sizes [36], the WEBSOM for text analysis [17], and the IKASL algorithm [5] that addresses challenges in incremental unsupervised learning. Moreover, recently an important direction is the simplification of the SOM algorithm [2, 18, 35] for improving its speed and power-efficiency.

However, only a limited body of work has explored the plausibility of the SOM beyond its original metric vector space. In contrast to a metric vector space, a symbolic data space is a non-vectorial representation that possesses an internal variation and structure which must be taken into account in computations. Records in a symbolic dataset are not limited to a single value, for instance, each data point can be a hypercube in p-dimensional space or Cartesian product of distribution. In [24], authors make the first effort to apply SOM algorithm to

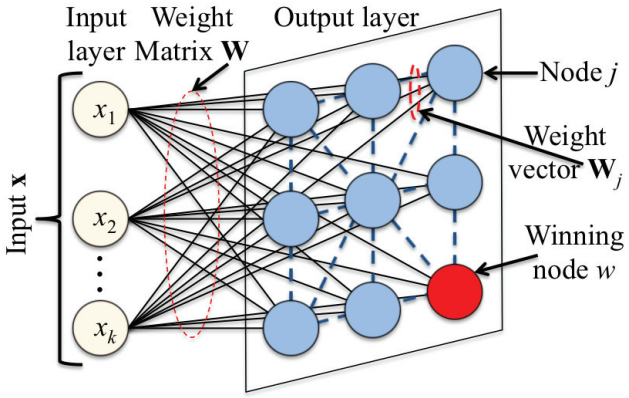


Fig. 3. Illustration of a Self-Organizing Map with nine nodes organized according to the grid topology.

symbol strings, the primary challenges were the discrete nature of data points and adjustments required for the learning rule, addressed using the generalized means/medians and batch map principle. Research reported in [38] takes a more direct approach to n-gram modeling of HTTP requests from network logs. Feature matrices are formed by counting the occurrences of n-characters corresponding to each array in the HTTP request, generating a memory-intensive feature vector of length 256^n. Feature matrices are fed into a variant of the SOM, Growing Hierarchical SOMs [9] to detect anomalous requests. Authors report both accuracy and precision of 99.9% on average, when using bigrams and trigrams. Given the limited awareness and availability of research into unsupervised machine learning on symbolic data, coupled with the increasing complexity of raw data [25], it is pertinent to investigate the functional synergies between hyperdimensional computing and the principles of SOMs.

3 Methods

This section presents the methods used in this paper. We describe: the basics of the SOM algorithm; the process of collecting n-gram statistics; the basics of hyperdimensional computing; and the mapping of n-gram statistics to the distributed representation using hyperdimensional computing.

3.1 Self-Organizing Maps

A SOM [23] (see Fig. 3) consists of a set of nodes arranged in a certain topology (e.g., a rectangular or a hexagonal grid or even a straight line). Each node j is characterized by a weight vector of dimensionality equal the dimensionality of an input vector (denoted as $\mathbf{x}$). The weight vectors are typically initialized at random. Denote a $u \times k$ matrix of k-dimensional weight vectors of u nodes in a

SOM as $\mathbf{W}$. Also denote a weight vector of node j as $\mathbf{W}_j$ and i'th positions of this vector as $\mathbf{W}_{ji}$. One of the main steps in the SOM algorithm is for a given input vector $\mathbf{x}$ to identify the the winning node, which has the closest weight vector to $\mathbf{x}$. Computation of a distance between the input $\mathbf{x}$ and the weight vectors in $\mathbf{W}$, the winner takes all procedure as well as the weight update rule are the main components of SOM logic. They are outlined in the text below.

In order to compare $\mathbf{x}$ and $\mathbf{W}_j$, a similarity measure is needed. The SOM uses Euclidian distance:

$$D(\mathbf{x}, \mathbf{W}_j) = \sqrt{\sum_{i=1}^{i=k} (\mathbf{x}_i - \mathbf{W}_{ji})^2}, \tag{1}$$

where $\mathbf{x}_i$ and $\mathbf{W}_{ji}$ are the corresponding values of ith positions. The winning node (denoted as w) is defined as a node with the lowest Euclidian distance to the input $\mathbf{x}$.

In the SOM, a neighborhood $\mathcal{M}$ of nodes around the winning node w is selected and updated; the size of the neighborhood progressively decreases:

$$\gamma(j, w, t) = e^{-l(j,w)/2\sigma(t)^2}, \tag{2}$$

where $l(j, w)$ is the lateral distance between a node j and the winning node w on the SOM's topology; $\sigma(t)$ is the decreasing function, which depends of the current training iteration t. If a node j is within the neighborhood $\mathcal{M}$ of w then the weight vector $\mathbf{W}_j$ is updated with:

$$\triangle \mathbf{W}_j = \eta(t)\gamma(j, w, t)(\mathbf{x} - \mathbf{W}_j), \tag{3}$$

where $\eta(t)$ denotes the learning rate decreasing with increasing t. During an iteration t, the weights are updated for all available training inputs $\mathbf{x}$. The training process usually runs for T iterations.

Once the SOM has been trained it could be used in the operating phase. The operating phase is very similar to that of the training one except that the weights stored in $\mathbf{W}$ are kept fixed. For a given input $\mathbf{x}$, the SOM identifies the winning node w. This information is used depending on the task at hand. For example, in clustering tasks, a node could be associated with a certain region. In this paper, we consider the classification task, and therefore, each node would have an assigned classification label.

3.2 n-gram statistics

In order to calculate n-gram statistics for the input symbolic data $\mathcal{D}$, which is described by the alphabet of size a, we first initialize an empty vector $\mathbf{s}$. This vector will store the n-gram statistics for $\mathcal{D}$, where the ith position in $\mathbf{s}$ corresponds to an n-gram $\boldsymbol{\mathcal{N}}_i = \langle \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n, \rangle$ from the set $\boldsymbol{\mathcal{N}}$ of all unique n-grams; $\mathcal{S}_j$ corresponds to a symbol in jth position of $\boldsymbol{\mathcal{N}}_i$. The value $\mathbf{s}_i$ indicates the number of times $\boldsymbol{\mathcal{N}}_i$ was observed in the input symbolic data $\mathcal{D}$. The dimensionality of

$\mathbf{s}$ is equal to the total number of n-grams in $\boldsymbol{\mathcal{N}}$, which in turn depends on a and n (size of n-grams) and is calculated as a^n (i.e., $\mathbf{s} \in [a^n \times 1]$). The n-gram statistics $\mathbf{s}$ is calculated via a single pass through $\mathcal{D}$ using the overlapping sliding window of size n, where for an n-gram observed in the current window the value of its corresponding position in $\mathbf{s}$ (i.e., counter) is incremented by one. Thus, $\mathbf{s}$ characterizes how many times each n-gram in $\boldsymbol{\mathcal{N}}$ was observed in $\mathcal{D}$.

3.3 Hyperdimensional computing

Hyperdimensional computing [16, 29, 31] also known as Vector Symbolic Architectures is a family of bio-inspired methods of representing and manipulating concepts and their meanings in a high-dimensional space. Hyperdimensional computing finds its applications in, for example, cognitive architectures [10], natural language processing [34], biomedical signal processing [20], approximation of conventional data structures [21,28], and for classification tasks, such as gesture recognition [22] [4], physical activity recognition [33], fault isolation [19]. Vectors of high (but fixed) dimensionality (denoted as d) are the basis for representing information in hyperdimensional computing. These vectors are often referred to as high-dimensional vectors or HD vectors. The information is distributed across HD vectors positions, therefore, HD vectors use distributed representations. Distributed representations [13] are contrary to the localist representations (which are used in the conventional n-gram statistics) since any subset of the positions can be interpreted. In other words, a particular position of an HD vector does not have any interpretable meaning – only the whole HD vector can be interpreted as a holistic representation of some entity, which in turn bears some information load. In the scope of this paper, symbols of the alphabet are the most basic components of a system and their atomic HD vectors are generated randomly. Atomic HD vectors are stored in the so-called item memory, which in its simplest form is a matrix. Denote the item memory as $\mathbf{H}$, where $\mathbf{H} \in [d \times a]$. For a given symbol $\mathcal{S}$ its corresponding HD vector from $\mathbf{H}$ is denoted as $\mathbf{H}_{\mathcal{S}}$. Atomic HD vectors in $\mathbf{H}$ are bipolar ($\mathbf{H}_{\mathcal{S}} \in \{-1, +1\}^{[d \times 1]}$) and random with equal probabilities for $+1$ and -1. It is worth noting that an important property of high-dimensional spaces is that with an extremely high probability all random HD vectors are dissimilar to each other (quasi orthogonal).

In order to manipulate atomic HD vectors hyperdimensional computing defines operations and a similarity measure on HD vectors. In this paper, we use the cosine similarity for characterizing the similarity. Three key operations for computing with HD vectors are bundling, binding, and permutation.

The binding operation is used to bind two HD vectors together. The result of binding is another HD vector. For example, for two symbols $\mathcal{S}_1$ and $\mathcal{S}_2$ the result of binding of their HD vectors (denotes as $\mathbf{b}$) is calculated as follows:

$$\mathbf{b} = \mathbf{H}_{\mathcal{S}_1} \odot \mathbf{H}_{\mathcal{S}_2}, \tag{4}$$

where the notation $\odot$ for the Hadamard product is used to denote the binding operation since this paper uses positionwise multiplication for binding. An important property of the binding operation is that the resultant HD vector $\mathbf{b}$ is

dissimilar to the HD vectors being bound, i.e., the cosine similarity between $\mathbf{b}$ and $\mathbf{H}_{\mathcal{S}_1}$ or $\mathbf{H}_{\mathcal{S}_2}$ is approximately 0.

An alternative approach to binding when there is only one HD vector is to permute (rotate) the positions of the HD vector. It is convenient to use a fixed permutation (denoted as ρ) to bind a position of a symbol in a sequence to an HD vector representing the symbol in that position. Thus, for a symbol $\mathcal{S}_1$ the result of permutation of its HD vector (denotes as $\mathbf{r}$) is calculated as follows:

$$\mathbf{r} = \rho(\mathbf{H}_{\mathcal{S}_1}). \tag{5}$$

Similar to the binding operation, the resultant HD vector $\mathbf{r}$ is dissimilar to $\mathbf{H}_{\mathcal{S}_1}$.

The last operation is called bundling. It is denoted with + and implemented via positionwise addition. The bundling operation combines several HD vectors into a single HD vector. For example, for $\mathcal{S}_1$ and $\mathcal{S}_2$ the result of bundling of their HD vectors (denotes as $\mathbf{a}$) is simply:

$$\mathbf{a} = \mathbf{H}_{\mathcal{S}_1} + \mathbf{H}_{\mathcal{S}_2}. \tag{6}$$

In contrast to the binding and permutation operations, the resultant HD vector $\mathbf{a}$ is similar to all bundled HD vectors, i.e., the cosine similarity between $\mathbf{b}$ and $\mathbf{H}_{\mathcal{S}_1}$ or $\mathbf{H}_{\mathcal{S}_1}$ is more than 0. Thus, the bundling operation allows storing information in HD vectors [11]. Moreover if several copies of any HD vector are included (e.g., $\mathbf{a} = 3\mathbf{H}_{\mathcal{S}_1} + \mathbf{H}_{\mathcal{S}_2}$), the resultant HD vector is more similar to the dominating HD vector than to other components.

3.4 Mapping of n-gram statistics with hyperdimensional computing

The mapping of n-gram statistics into distributed representation using hyperdimensional computing was first shown in [15]. At the initialization phase, the random item memory $\mathbf{H}$ is generated for the alphabet. A position of symbol $\mathcal{S}_j$ in $\boldsymbol{\mathcal{N}}_i$ is represented by applying the fixed permutation ρ to the corresponding atomic HD vector $\mathbf{H}_{\mathcal{S}_j}$ j times, which is denoted as $\rho^j(\mathbf{H}_{\mathcal{S}_j})$. Next, a single HD vector for $\boldsymbol{\mathcal{N}}_i$ (denoted as $\mathbf{m}_{\boldsymbol{\mathcal{N}}_i}$) is formed via the consecutive binding of permuted HD vectors $\rho^j(\mathbf{H}_{\mathcal{S}_j})$ representing symbols in each position j of $\boldsymbol{\mathcal{N}}_i$. For example, for the trigram 'cba' will be mapped to its HD vector as follows: $\rho^1(\mathbf{H}_{\mathrm{c}}) \odot \rho^2(\mathbf{H}_{\mathrm{b}}) \odot \rho^3(\mathbf{H}_{\mathrm{a}})$. In general, the process of forming HD vector of an n-gram can be formalized as follows:

$$\mathbf{m}_{\boldsymbol{\mathcal{N}}_i} = \prod_{j=1}^{n} \rho^j(\mathbf{H}_{\mathcal{S}_j}), \tag{7}$$

where $\prod$ denotes the binding operation (positionwise multiplication) when applied to n HD vectors.

Once it is known how to map a particular n-gram to an HD vector, mapping the whole n-gram statistics $\mathbf{s}$ is straightforward. HD vector $\mathbf{h}$ corresponding to

$\mathbf{s}$ is created by bundling together all n-grams observed in the data, which is expressed as follows:

$$\mathbf{h} = \sum_{i=1}^{a^n} \mathbf{s}_i \mathbf{m}_{\mathcal{N}_i} = \sum_{i=1}^{a^n} \mathbf{s}_i \prod_{j=1}^{n} \rho^j(\mathbf{H}_{\mathcal{S}_j}), \tag{8}$$

where $\sum$ denotes the bundling operation when applied to several HD vectors. Note that $\mathbf{h}$ is not bipolar, therefore, in the experiments below we normalized it by its ℓ_2 norm.

4 Experimental results

This section describes the experimental results studying several configurations of the proposed approach and comparing it with the results obtained for the conventional n-gram statistics. We slightly modified the experimental setup from that used in [15], where the task was to identify a language of a given text sample (i.e., for a string of symbols). The language recognition was done for 21 European languages. The list of languages is as follows: Bulgarian, Czech, Danish, German, Greek, English, Estonian, Finnish, French, Hungarian, Italian, Latvian, Lithuanian, Dutch, Polish, Portuguese, Romanian, Slovak, Slovene, Spanish, Swedish. The training data is based on the Wortschatz Corpora [30]. The average size of a language's corpus in the training data was $1,085,637.3 \pm 121,904.1$ symbols. It is worth noting, that in the experiments reported in [15] the whole training corpus of a particular language was used to estimate the corresponding n-grams statistics. While in this study, in order to enable training of SOMs, each language corpus was divided into samples where the length of each sample was set to $1,000$ symbols. The total number of samples in the training data was $22,791$. The test data is based on the Europarl Parallel Corpus[4]. The test data also represent 21 European languages. The total number of samples in the test data was $21,000$, where each language was represented with $1,000$ samples. Each sample in the test data corresponds to a single sentence. The average size of a sample in the test data was 150.3 ± 89.5 symbols.

The data for each language was preprocessed such that the text included only lower case letters and spaces. All punctuation was removed. Lastly, all text used the 26-letter ISO basic Latin alphabet, i.e., the alphabet for both training and test data was the same and it included 27 symbols. For each text sample the n-gram statistics (either conventional or mapped to the distributed representation) was obtained, which was then used as input $\mathbf{x}$ when training or testing SOMs. Since each sample was preprocessed to use the alphabet of only $a = 27$ symbols, the conventional n-gram statistics input is 27^n dimensional (e.g., $k = 729$ when $n = 2$) while the dimensionality of the mapped n-gram statistics depends on the dimensionality of HD vectors d (i.e., $k = d$). In all experiments reported in this paper, we used the standard SOMs implementation, which is a part of the Deep Learning Toolbox in MATLAB R2018B (Mathworks Inc, Natick, Ma.)

[4] Available online at http://www.statmt.org/europarl/.

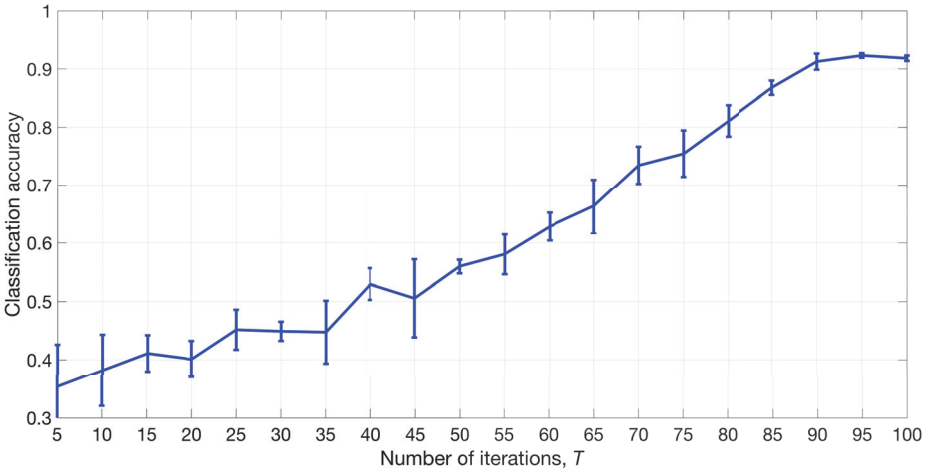


Fig. 4. The classification accuracy of the SOM trained on the conventional bigram statistics ($n = 2$; $k = 729$) against the number of training iterations T. The grid size was set to ten ($u = 100$). T varied in the range $[5, 100]$ with step 5.

During the experiments, certain parameters SOM were fixed. In particular, the topology of SOMs was set to the standard grid topology. The initial size of the neighborhood was always fixed to ten. The size of the neighborhood and the learning rate were decreasing progressively with training according to the default rules of the used implementation. In all simulations, a SOM was trained for a given number of iterations T, which was set according to an experiment reported in Fig. 4. All reported results were averaged across five independent simulations. The bars in the figure show standard deviations.

Recall that SOMs are suited for the unsupervised training, therefore, an extra mechanism is needed to use them in supervised tasks such as the considered language recognition task, i.e., once the SOM is trained there is still a need to assign a label to each trained node. After training a SOM for T iterations using all $22,791$ training samples, the whole training data were presented to the trained SOM one more time without modifying $\mathbf{W}$. Labels for the training data were used to collect the statistics for the winning nodes. The nodes were assigned the labels of the languages dominating in the collected statistics. If a node in the trained SOM was never chosen as the winning node for the training samples (i.e., its statistics information is empty) then this node was ignored during the testing phase. During the testing phase, $21,000$ samples of the test data were used to assess the trained SOM. For each sample in the test data, the winning node was determined. The test sample then was assigned the language label corresponding to its winning node. The classification accuracy was calculated using the SOM predictions and the ground truth of the test data. The accuracy was used as the main performance metric for evaluation and comparison of different SOMs. It is worth emphasizing that the focus of experiments is not on achieving the highest

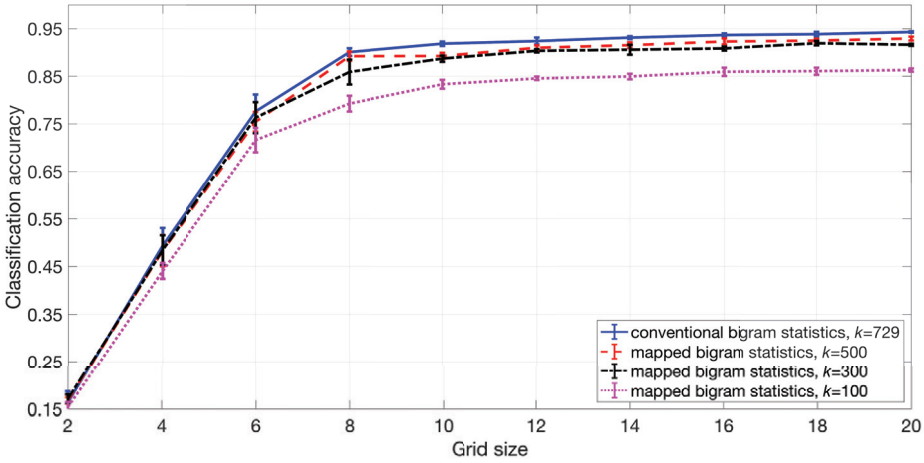


Fig. 5. The classification accuracy of the SOM against the grid size for the case of bigram statistics. The grid size varied in the range $[2, 20]$ with step 2.

possible accuracy but on a comparative analysis of SOMs with the conventional n-gram statistics versus SOMs with the mapped n-gram statistics with varying d. However, it is worth noting that the accuracy, obtained when collecting an n-gram statistics profile for each language $[15, 32]$ for $n = 2$ and $n = 3$ and using the nearest neighbor classifier, was 0.945 and 0.977 respectively. Thus, the results presented below for SOMs match the ones obtained with the supervised learning on bigrams when the number of nodes is sufficiently high. In the case of trigrams, the highest accuracy obtained with SOMs was slightly (about 0.02) lower. While SOMs not necessarily achieve the highest accuracy compared to the supervised methods, their important advantage is data visualization. For example, in the considered task one could imagine using the trained SOM for identifying the clusters typical for each language and even reflecting on their relative locations on the map.

The experiment in Fig. 4 presents the classification accuracy of the SOM trained on the conventional bigram statistics against T. The results demonstrated that the accuracy increased with the increased number T. Moreover, for higher values of T the predictions are more stable. The performance started to saturate at T more than 90, therefore, in the other experiments the value of T was fixed to 100.

The grid size varied in the range $[2, 20]$ with step 2, i.e, the number of nodes u varied between 4 and 400. In Fig. 5 the solid curve corresponds to the SOM trained on the conventional bigram statistics. The dashed, dash-dot, and dotted curves correspond to the SOMs trained on the mapped bigram statistics with $k = d = 500$, $k = d = 300$, and $k = d = 100$ respectively.

The experiment presented in Fig. 5 studied the classification accuracy of the SOM against the grid size for the case of bigram statistics. Note that the

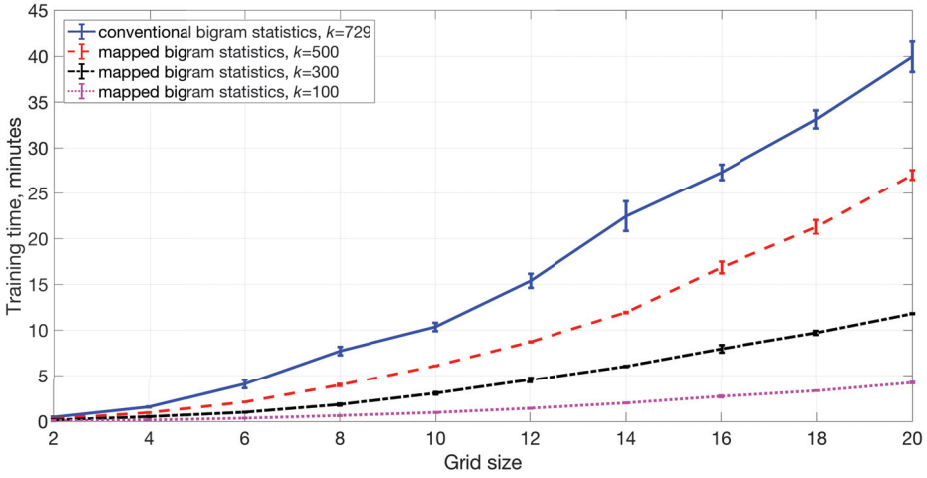


Fig. 6. The training time of the SOM against the grid size for the case of bigram statistics. The grid size varied in the range $[2, 20]$ with step 2.

number of nodes u in the SOM is proportional to the square of the grid size. For example, when the gris size equals 2 the SOM has $u = 4$ nodes while when it equals 20 the SOM has $u = 400$ nodes. The results in Fig. 5 demonstrated that the accuracy of all considered SOMs improves with the increased grid size. It is intuitive that all SOMs with grid sizes less than five performed poorly since the number of nodes in SOMs was lower than the number of different languages in the task. Nevertheless, the performance of all SOMs was constantly improving with the increased grid size, but the accuracy started to saturate at about 100 nodes. Moreover, increasing the dimensionality of HD vectors d was improving the accuracy. Note, however, that there was a better improvement when going from $d = 100$ to $d = 300$ compared to increasing d from 300 to 500. The performance of the conventional bigram statistics was already approximated well even when $d = 300$; for $d = 500$ the accuracy was just slightly worse than that of the conventional bigram statistics.

It is important to mention that the usage of the mapped n-grams statistics allows decreasing the size of $\mathbf{W}$ in proportion to d/a^n. Moreover, it allows decreasing the training time of SOMs. The experiment in Fig. 6 presents the training time of the SOM against the grid size for the case of bigram statistics. Fig. 6 corresponds to that of Fig. 5. The number of training iterations was fixed to $T = 100$. For example, for grid size 16 the average training time on a laptop for $k = d = 100$ was 2.7 minutes (accuracy 0.86); for $k = d = 300$ it was 8.0 minutes (accuracy 0.91); for $k = d = 500$ it was 16.9 minutes (accuracy 0.92); and for $k = a^n = 729$ it was 27.3 minutes (accuracy 0.93). Thus, the usage of the mapping allows the trade-off between the obtained accuracy and the required computational resources.

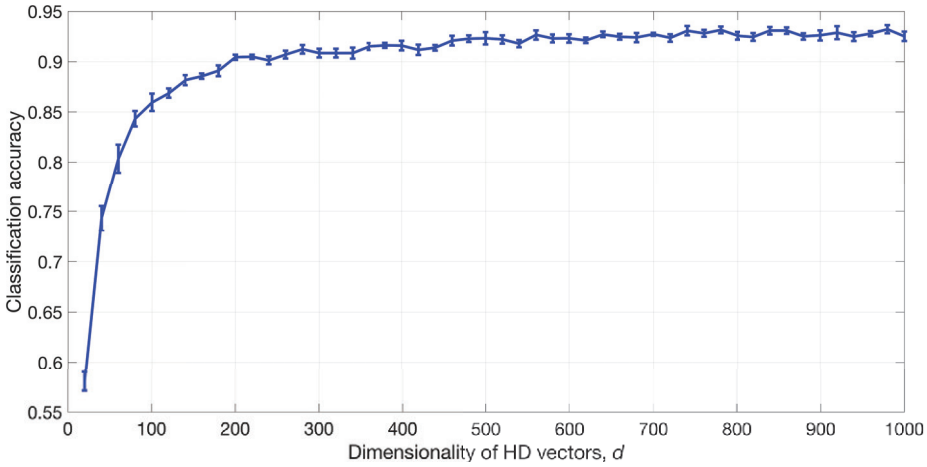


Fig. 7. The classification accuracy of the SOM trained on the mapped bigram statistics ($n = 2$) against the dimensionality of HD vectors d ($k = d$). The grid size was set to 16 ($u = 256$). The number of training iterations T was fixed to 100.

In order to observe a more detailed dependency between the classification accuracy and the dimensionality of distributed representations d of the mapped n-gram statistics, an additional experiment was done. Fig. 7 depicts the results. The dimensionality of distributed representations d varied in the range $[20, 1000]$ with step 20. It is worth mentioning that even for small dimensionalities ($d < 100$), the accuracy is far beyond random. The results in Fig. 7 are consistent with the observations in Fig. 5 in a way that the accuracy was increasing with the increased d. The performance saturation begins for the values above 200 and the improvements beyond $d = 500$ look marginal. Thus, we experimentally observed that the quality of mappings grows with d, however, after a certain saturation point increasing d further becomes impractical.

The last experiment in Fig. 8 is similar to Fig. 5 but it studied the classification accuracy for the case of trigram statistics ($n = 3$). The grid size varied in the range $[2, 20]$ with step 2. The solid curve corresponds to the SOM trained on the conventional trigram statistics ($k = 27^3 = 19,683$). The dashed and dash-dot curves correspond to the SOMs trained on the mapped trigram statistics with $k = d = 5,000$ and $k = d = 1,000$ respectively. The results in Fig. 8 are consistent with the case of bigrams. The classification of SOMs was better for higher d and even when $d < a^n$ the accuracy was approximated well.

5 Conclusions

This paper presented an approach for the mapping of n-gram statistics into vectors of fixed arbitrary dimensionality, which does not depend on the size of n-grams n. The mapping is aided by hyperdimensional computing a bio-

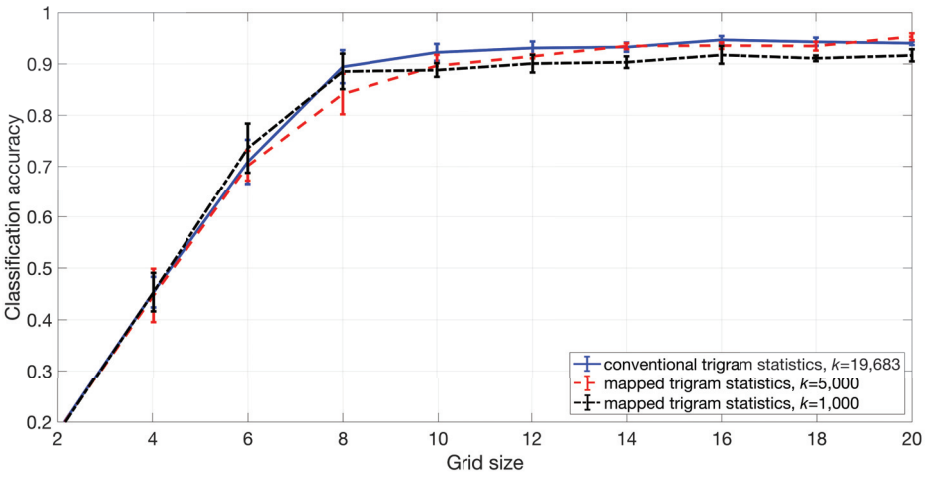


Fig. 8. The classification accuracy of the SOM against the grid size for the case of trigram statistics ($n = 3$). The number of training iterations T was fixed to 100.

inspired approach for computing with large random vectors. Mapped in this way n-gram statistics is used as the input to Self-Organized Maps. This novel for Self-Organized Maps step allows removing the computational bottleneck caused by the exponentially growing dimensionality of n-gram statistics with increased n. While preserving the performance of the trained Self-Organized Maps (as demonstrated in the languages recognition task) the presented approach results in reduced memory consumption due to smaller weight matrix (proportional to d and u) and shorter training times. The main limitation of this study is that we have validated the proposed approach only on a single task when using the conventional Self-Organized Maps. However, it is worth noting that the proposed approach could be easily used for other modifications of the conventional Self-Organizing Maps such as Growing Self-Organizing Maps [1], where dynamic topology preservation facilitates unconstrained learning. This is in contrast to a fixed-structure feature map as the map itself is defined by the unsupervised learning process of the feature vectors. We intend to investigate distributed representation of n-gram statistics in structure-adapting feature maps in future work.

References

1. Alahakoon, D., Halgamuge, S., Srinivasan, B.: Dynamic Self-Organizing Maps with Controlled Growth for Knowledge Discovery. IEEE Transactions on Neural Networks **11**(3), 601–614 (2000)
2. Appiah, K., Hunter, A., Dickinson, P., Meng, H.: Implementation and Applications of Tri-State Self-Organizing Maps on FPGA. IEEE Transactions on Circuits and Systems for Video Technology **22**(8), 1150–1160 (2012)

3. Bandaragoda, T.R., De Silva, D., Alahakoon, D.: Automatic Event Detection in Microblogs using Incremental Machine Learning. Journal of the Association for Information Science and Technology **68**(10), 2394–2411 (2017)
4. Bishop, C.M., Svensén, M., Williams, C.K.: GTM: The Generative Topographic Mapping. Neural computation **10**(1), 215–234 (1998)
5. De Silva, D., Alahakoon, D.: Incremental Knowledge Acquisition and Self Learning from Text. In: International Joint Conference on Neural Networks (IJCNN). pp. 1–8. IEEE (2010)
6. De Silva, D., Alahakoon, D., Yu, X.: A Data Fusion Technique for Smart Home Energy Management and Analysis. In: Annual Conference of the IEEE Industrial Electronics Society (IECON). pp. 4594–4600 (2016)
7. De Silva, D., Ranasinghe, W., Bandaragoda, T., Adikari, A., Mills, N., Iddamalgoda, L., Alahakoon, D., Lawrentschuk, N., Persad, R., Osipov, E., Gray, R., Bolton, D.: Machine Learning to Support Social Media Empowered Patients in Cancer Care and Cancer Treatment Decisions. PloS One **13**(10), 1–10 (2018)
8. De Silva, D., Yu, X., Alahakoon, D., Holmes, G.: A Data Mining Framework for Electricity Consumption Analysis from Meter Data. IEEE Transactions on Industrial Informatics **7**(3), 399–407 (2011)
9. Dittenbach, M., Merkl, D., Rauber, A.: The Growing Hierarchical Self-Organizing Map. In: International Joint Conference on Neural Networks (IJCNN). vol. 6, pp. 15–19 (2000)
10. Eliasmith, C.: How to Build a Brain. Oxford University Press (2013)
11. Frady, E.P., Kleyko, D., Sommer, F.T.: A Theory of Sequence Indexing and Working Memory in Recurrent Neural Networks. Neural Computation **30**, 1449–1513 (2018)
12. Hazan, H., Saunders, D.J., Sanghavi, D.T., Siegelmann, H.T., Kozma, R.: Unsupervised Learning with Self-Organizing Spiking Neural Networks. In: International Joint Conference on Neural Networks (IJCNN). pp. 1–6 (2018)
13. Hinton, G., McClelland, J., Rumelhart, D.: Distributed Representations. In: Rumelhart, D., McClelland, J. (eds.) Parallel Distributed Processing. Explorations in the Microstructure of Cognition. Volume 1. Foundations. pp. 77–109. MIT Press (1986)
14. Jayaratne, M., Alahakoon, D., De Silva, D., Yu, X.: Bio-Inspired Multisensory Fusion for Autonomous Robots. In: Annual Conference of the IEEE Industrial Electronics Society (IECON). pp. 3090–3095 (2018)
15. Joshi, A., Halseth, J., Kanerva, P.: Language Geometry Using Random Indexing. In: Quantum Interaction (QI). pp. 265–274 (2016)
16. Kanerva, P.: Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. Cognitive Computation **1**(2), 139–159 (2009)
17. Kaski, S., Honkela, T., Lagus, K., Kohonen, T.: WEBSOM–Self-organizing maps of document collections1. Neurocomputing **21**(1-3), 101–117 (1998)
18. Kleyko, D., Osipov, E., De Silva, D., Wiklund, U., Alahakoon, D.: Integer Self-Organizing Maps for Digital Hardware. In: International Joint Conference on Neural Networks (IJCNN). pp. 1–8 (2019)
19. Kleyko, D., Osipov, E., Papakonstantinou, N., Vyatkin, V.: Hyperdimensional Computing in Industrial Systems: The Use-Case of Distributed Fault Isolation in a Power Plant. IEEE Access **6**, 30766–30777 (2018)
20. Kleyko, D., Osipov, E., Wiklund, U.: A Hyperdimensional Computing Framework for Analysis of Cardiorespiratory Synchronization During Paced Deep Breathing. IEEE Access **7**, 34403–34415 (2019)

21. Kleyko, D., Rahimi, A., Gayler, R., Osipov, E.: Autoscaling Bloom Filter: Controlling Trade-off Between True and False Positives. arXiv:1705.03934 pp. 1–13 (2017)
22. Kleyko, D., Rahimi, A., Rachkovskij, D., Osipov, E., Rabaey, J.: Classification and Recall with Binary Hyperdimensional Computing: Trade-offs in Choice of Density and Mapping Characteristic. IEEE Transactions on Neural Networks and Learning Systems **29**(12), 5880–5898 (2018)
23. Kohonen, T.: Self-Organizing Maps. Springer Series in Information Sciences (2001)
24. Kohonen, T., Somervuo, P.: Self-Organizing Maps of Symbol Strings. Neurocomputing **21**(1-3), 19–30 (1998)
25. Kusiak, A.: Smart Manufacturing must Embrace Big Data. Nature News **544**(7648), 23 (2017)
26. Nallaperuma, D., De Silva, D., Alahakoon, D., Yu, X.: Intelligent Detection of Driver Behavior Changes for Effective Coordination Between Autonomous and Human Driven Vehicles. In: Annual Conference of the IEEE Industrial Electronics Society (IECON). pp. 3120–3125 (2018)
27. Nawaratne, R., Bandaragoda, T., Adikari, A., Alahakoon, D., De Silva, D., Yu, X.: Incremental Knowledge Acquisition and Self-Learning for Autonomous Video Surveillance. In: Annual Conference of the IEEE Industrial Electronics Society (IECON). pp. 4790–4795 (2017)
28. Osipov, E., Kleyko, D., Legalov, A.: Associative Synthesis of Finite State Automata Model of a Controlled Object with Hyperdimensional Computing. In: Annual Conference of the IEEE Industrial Electronics Society (IECON). pp. 3276–3281 (2017)
29. Plate, T.A.: Holographic Reduced Representations: Distributed Representation for Cognitive Structures. Stanford: Center for the Study of Language and Information (CSLI) (2003)
30. Quasto, U., Richter, M., Biemann, C.: Corpus Portal for Search in Monolingual Corpora. In: Fifth International Conference on Language Resources and Evaluation (LREC). pp. 1799–1802 (2006)
31. Rahimi, A., Datta, S., Kleyko, D., Frady, E.P., Olshausen, B., Kanerva, P., Rabaey, J.M.: High-dimensional Computing as a Nanoscalable Paradigm. IEEE Transactions on Circuits and Systems I: Regular Papers **64**(9), 2508–2521 (2017)
32. Rahimi, A., Kanerva, P., Rabaey, J.: A Robust and Energy Efficient Classifier Using Brain-Inspired Hyperdimensional Computing. In: IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED). pp. 64–69 (2016)
33. Rasanen, O., Kakouros, S.: Modeling Dependencies in Multiple Parallel Data Streams with Hyperdimensional Computing. IEEE Signal Processing Letters **21**(7), 899–903 (2014)
34. Recchia, G., Sahlgren, M., Jones, P.K.M.: Encoding Sequential Information in Semantic Space Models. Comparing Holographic Reduced Representation and Random Permutation. Computational Intelligence and Neuroscience pp. 1–18 (2015)
35. Santana, A., Morais, A., Quiles, M.: An Alternative Approach for Binary and Categorical Self-Organizing Maps. In: International Joint Conference on Neural Networks (IJCNN). pp. 2604–2610 (2017)
36. Shah-Hosseini, H., Safabakhsh, R.: TASOM: a New Time Adaptive Self-Organizing Map. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **33**(2), 271–282 (2003)
37. Vesanto, J., Alhoniemi, E.: Clustering of the Self-Organizing Map. IEEE Transactions on Neural Networks **11**(3), 586–600 (2000)

38. Zolotukhin, M., Hamalainen, T., Juvonen, A.: Online Anomaly Detection by using n-gram Model and Growing Hierarchical Self-Organizing Maps. In: 8th International Wireless Communications and Mobile Computing Conference (IWCMC). pp. 47–52 (2012)

Use of color for arrangement of web publication of science news on the corporate site of the Siberian Branch of Russian Academy of Sciences

Olga A. Klimenko

Institute of Computational Technologies SB RAS, Academician M.A. Lavrentiev avenue, 6, 630090, Novosibirsk, Russia

klimenko@ict.nsc.ru

Abstract. The corporate site of the Siberian Branch of the Russian Academy of Sciences (SB RAS) www.sbras.ru has been operating since 1996 and represents all aspects of the activities of a research corporation. The SB RAS provides scientific and methodological guidance to 86 research organizations in the fields of mathematics, mechanics, physics, chemistry, biology, geology, medicine, economic and humanities, Computer Science, agricultural sciences. The corporate website publishes daily news from institutes and research centers about basic and applied research, conferences, symposia and seminars. On the news feed of the site and in the SB RAS Database for news and research organizations are associated with a certain color. For example, all news related to medical research is colored red, economic news is colored yellow, energy news presented in orange. Each month it is calculated how much news was presented on the topic "Physical Sciences", "Chemical Sciences", "Humanities", etc. Using this data, an infographic www.sbras.ru/en/about is created. Using color to systematize scientific information makes it easier to identify trends.

Keywords: Database and information systems, Collections and archives, Web publishing.

1 Introduction

Siberian Branch of the Russian Academy of Sciences (SB RAS) is the largest integrator and the main expert of research and development, scientific educational, experimental design and industrial organizations in the Eastern Russia. SB RAS was established in 1957. The initiative originated from Mikhail Lavrentyev, Sergei Sobolev and Sergei Khristianovich. Mikhail Lavrentyev became its founding chairman. The SB RAS provides scientific and methodological guidance to 86 research organizations in the fields of mathematics, mechanics, physics, chemistry, biology, geology, medicine, economic and humanities, Computer Science, agricultural sciences. The central location of the Siberian Branch is Novosibirsk. Research centers are established in Novosibirsk, Tomsk, Krasnoyarsk, Irkutsk, Yakutsk, Ulan-Ude, Kemerovo, Tyumen, and

Omsk. Some of the research institutions are located in Barnaul, Biisk, Chita and Kyzyl.

The corporate site “Portal SB RAS” was created in 1996 by members of the Institute of Computational Technologies SB RAS. Since its inception, Portal SB RAS has expanded due to additional information systems. Portal SB RAS includes databases of projects, conferences, scientific organizations. Annual reports of the SB RAS since 1995, and other important documents of the corporation are stored on the corporate website. For example, the database of documents stores decisions on the organization of new scientific institutions, information on appointments to the positions of directors of institutes. The Siberian Branch of the Russian Academy of Sciences publishes a weekly corporate newspaper. The archive contains electronic copies of newspaper issues since 1961.

Since 1996, the technological platforms on which the site was created have changed repeatedly. Archive databases and archive systems have been preserved and become part of the new site.

2 Webspace of the Siberian Branch of Russian Academy of Sciences

Portal SB RAS, the sites of the Siberian institutes and universities are combined by numerous links and form the academic webspace of the SB RAS (see Fig. 1).

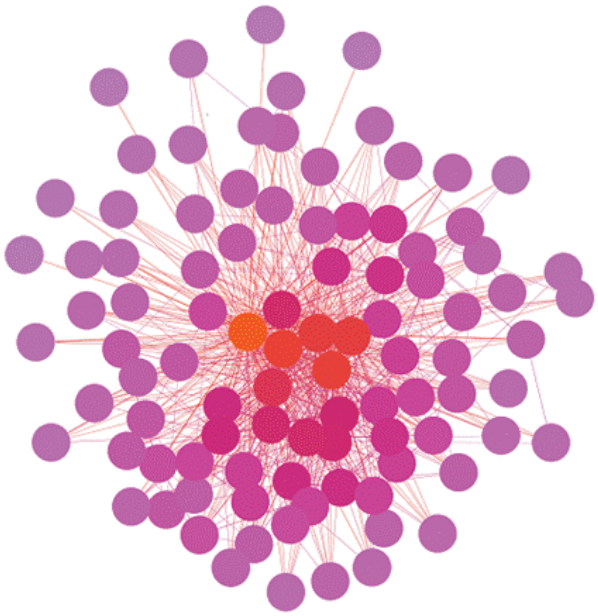


Fig. 1. Academic webspace of the SB RAS. Sites of institutes and Portal SB RAS are represented by circles, they are connected by links.

Academic webspace can be studied using webometric methods, special programs, methods of graph theory [1, 2]. This academic webspace of SB RAS was studied using a special program - a crawler, which examined links between sites. The program has found that the largest number of inbound and outbound links from sites of institutes contains Portal SB RAS. From all the sites of Siberian institutes there are links to the Portal SB RAS and vice versa. There is a group of sites that are most related. This group includes sites of the Institute of Computational Technologies SB RAS, A.P. Ershov Institute of Informatics Systems SB RAS, Sobolev Institute of Mathematics SB RAS, Institute of Cytology and Genetics SB RAS, State Public Scientific-Technological Library of the Siberian Branch of the RAS and a number of other sites. From the Siberian universities the Novosibirsk State University has the largest number of links to the sites of the institutes and Portal SB RAS.

The webspace of the SB RAS is similar to the webspace of the Fraunhofer Society in Germany and academic webspace of the Republic of Serbia [3, 4]. Academic communities have a central site that is linked to all other sites. A distinctive feature of the webspace of the SB RAS is the presence of a large number of links between institutions that are engaged in different areas of science. For example, there are links between the sites of mathematical and humanitarian institutes, between physical and chemical.

The webspace of the SB RAS has clusters within itself; in clusters, the connections between sites are closer. A physical and mathematical cluster was found, which includes institutes that deal with mathematics, mechanics, physics, and computer science. Another big cluster is the chemical and biological one, which unites institutes that specialize in the field of biology, genetics, chemistry, agriculture and medicine.

3 New corporate website of the Siberian Branch of the Russian Academy of Sciences with adaptive design

In 2016, a new corporate website of the Siberian Branch of the Russian Academy of Sciences was created using freely distributed software on the Drupal platform [5]. The new technological platform of Portal SB RAS has made it convenient to view the site from mobile devices (see Fig. 2).

The site contains two databases, in one stores information about organizations and employees [6], and documents that can be submitted on the site's news feed are stored in another database.

To add a document to the Database, you need to select a subject from the list:

Mathematics and Computer Science
Physical sciences
Nanotechnology and information technology
Energy, engineering, mechanics and control processes
Chemical sciences
Biological Sciences
Earth Sciences

Humanitarian sciences
Economics
Medical sciences
Agricultural Sciences
NSU - SB RAS
Other categories

Fig. 2. Portal SB RAS. Section “The Siberian Branch, an overview”.

The classification features of the announcement on the news feed include "Competitions and Grants", "Conferences", "General Meetings of the SB RAS" and other events.

The following rubricator codes are indicated for official documents:

Constitution and laws of the Russian Federation
Decrees and orders of the President of the Russian Federation
Decisions and orders of the Government of the Russian Federation
Documents of federal ministries and departments
Documents of the Russian Academy of Sciences
Documents of the Novosibirsk Region and the Novosibirsk Scientific Center of the Siberian Branch of the Russian Academy of Sciences
Documents of the SB RAS
Agendas of the Presidium of the SB RAS

Required fields to fill out when publishing news are "Title", "Summary", "Detailed content", "Date of publication", "Event Date". To the text of the news, you can attach an illustration in the form of a photo or a graphic, a presentation of the report, a link to additional information. To determine the place of the news on the news tape indicates the weight of the news. Usually, urgent and important news is published first in the list of news in one day.

The database The organization and Employees [6] contains contact information for all scientific institutions. The Base includes information on the leadership of the institutes and members of the Russian Academy of Sciences. Contact information consists of business phone numbers, e-mail, postal address of the organization, website address.

4 Use of color on the news feed of the corporate website of the SB RAS

On the news feed of the site and in the SB RAS Database for news and research organizations are associated with a certain color. For example, all news related to medical research is colored red, economic news is colored yellow (golden), and energy news is shown in orange. Biological sciences and agriculture are represented by different shades of green. For the presentation of news in mathematics, information technology, nanotechnology and physics, shades of blue are used, from dark to turquoise. The color of the earth - brown is used to refer to geological news. For news related to two areas of science, mixed color is used. If the news relates to three or more topics, it is painted in gray. Search for news and documents can be carried out on one topic or two at a time, for example, to search for news that relate to physical, mathematical, or chemical and biological research. Each month it is calculated how much news was presented on the topic "Physical Sciences", "Chemical Sciences", "Humanities", etc., and infographics are used to visualize it [7].
In 2018, over 1800 news on Portal SB RAS were associated with a specific subject area of research, they were distributed as follows:

Nanotechnology and information technology - 16.6%
Physical Sciences - 15%
Biological sciences - 12.3%
Earth sciences - 10.9%
Chemical sciences - 9.5%
Power engineering, mechanical engineering, mechanics and control processes - 6.9%
Medical sciences - 6.8%
Economic Sciences - 6.5%
Mathematics and computer science - 6%
Humanities - 5.6%
Agricultural sciences - 3.9%

Analysis of the news related to the two topics showed that information technologies have become more widely used in the humanities, as well as in economic and agricultural sciences. In the past two years, news related to the biological, chemical and medical sciences are regularly published on the SB RAS corporate website. These areas of research are new and correspond to world trends. News analysis showed that there is a strong relationship between nanotechnology, information technology and physics. The least integrated were the agricultural and medical sciences, the humanities with physics and the humanities with chemistry.

5 Conclusions

The corporate website of the Siberian Branch of the Russian Academy of Sciences has accumulated a large amount of information about research conducted by Siberian scientists. A special feature of the SB RAS is a wide range of scientific research and integration between the sciences. The study of the academic web space of the SB RAS has allowed us to find several scientific communities that include mathematical, physical, chemical, biological and other institutions. News on interdisciplinary research are presented on Portal SB RAS. The use of color in the news by areas of science has given the visualization of the distribution of information on topics. Analysis of information for 2018 showed the use of digital technology in all sciences, in medicine, agriculture, economy.

6 Acknowledgment

The author is deeply grateful to Marina Filippova (A.P. Ershov Institute of Informatics Systems SB RAS), Elena Rychkov and Igor Shabalnikov (Institute of Computational Technologies SB RAS) for the highly qualified technical and information support of Portal SB RAS.

References

1. Kosyakov, D.V., Gus'kov, A.E., Bykhovtsev, E.S.: Russia's academic institutes as mirrored by webometrics. Herald of the Russian Academy of Sciences 86(6), 490-499 (2016).
2. Thelwall, M. A Web crawler design for data mining. Journal of Information Science. 27. 319–325 (2001)
3. Shokin, Yu.I., Vesnin, A.Yu., Dobrynin, A.A., Klimenko, O.A., Konstantinova, E.V., Rychkova, E.V., Savin, M.Yu. Studying of scientific web space by webometrics and graph theory methods. In: Proceedings on MIT-2013, pp. 629-639. University of Pristina, Kosovo (2014)
4. Dehmer, M., Dobrynin, A.A., Konstantinova, E.V., Vesnin A.Yu., Klimenko, O.A., Shokin, Yu I., Rychkova, E.V., Medvedev, A.N.: Analysis of Webspaces of the Siberian Branch of the Russian Academy of Sciences and the Fraunhofer-Gesellschaft. Information Technology in Industry 1(6), 1-6 (2018).
5. SB RAS Homepage, https://www.sbras.ru/en, last accessed 2019/02/01.

6. SB RAS Organizations and Employees https://www.sbras.ru/en/sbras/db, last accessed 2019/02/01.
7. SB RAS Statistics on the subject of news https://www.sbras.ru/en/about, last accessed 2019/02/01.

Rapid Instruction Decoding for IA-32

Yauhen Klimiankou

Department of Software for Information Technologies
Belarusian State University of Informatics and Radioelectronics
6 P. Brovki Street, Minsk 220013 Belarus
klimenkov@bsuir.by

Abstract. This paper explains new performance-oriented instruction decoder for IA-32 ISA. The decoder provides the functionality required for program analysis and interpretation and exports simple interface for conversion of code byte stream into a stream of generalized instruction descriptions. We report measurements comparing our decoder with well-known alternative solutions to demonstrate its superior efficiency.

Keywords: IA-32 · instructions decoding

1 Introduction

This paper attempts to shed light on an essential topic of design and implementation of efficient instruction decoders for CISC-like bytecodes. Importance of instruction decoders can be emphasized by the fact, that a wide range of applications including simulators [10], emulators [1], virtual machines [4], tools for static and dynamic analysis of executables [2, 5], disassemblers, decompilers [6], and others uses them. In the case of its usage in the area of simulators, emulators and virtual machines the decoder performance becomes one of the primary contributors to the efficiency of the entire system.

We draw attention to different approaches used for design and implementation of software decoders for complex CISC-like ISAs with variable instruction length. RISC-like ISAs usually assumes fixed instruction length and few easily parsable and distinguishable instruction formats. That leads to straightforward decoders both in hardware and in software implementations. Variable instruction length and variety of instruction formats pump significant complexity into decoder design and implementation in the case of CISC-like ISAs with respective degradation of performance.

We have developed a new instruction decoder for IA-32 ISA [9] which is a canonical example of CISC ISA. The decoder design focuses on applications in a broad range of domains including such as virtual machines and emulators for which instruction decoding efficiency is critical. For example, such projects as IBM PC compatible emulator Bosch [1] and hypervisor QEMU [4] can use it as front-end. The proposed decoder is well-abstracted from back-end logic, provides a pure interface and preserves universal nature in contrast to the original instruction decoders used in these projects.

We have explored various techniques and approaches towards optimization of instructions decoding performance. Our experience has shown that Mealy machine based decoder design leads to simple, flexible, extensible and efficient implementations. Our decoder demonstrates that there is a significant performance improvement which can be obtained by precaching of decoded instructions not containing variable part. Such instructions are most frequently faced instructions in industrial programs which amplifies the power of such performance trick.

Finally, we have compared the performance of our decoder implementation for IA-32 ISA with popular and extensively used analogs. We show that our decoder demonstrates its advantage in instructions decoding performance on real industrial quality binary program code.

Our key contributions in this work are:

- To present instruction decoder supporting CISC-like bytecode that reproduces IA-32 ISA.
- To explore design principles and optimization tricks towards efficient decoding of CISC-like bytecodes.
- To present the comparison between different instruction decoders for IA-32 ISA.
- To show that it is practical to use the design based on Mealy machine automata with use of table-guided dispatching, extensive precaching and simplified output interface.

2 IA-32 Instruction Set Architecture

IA-32, also known as *i386*, is a 32-bit version of the *x86* ISA introduced in 1985. Successive generations of microprocessors have extremely complicated *x86* ISA over years. "Manual for Intel 8086" (1979) [7] contains only 43 pages about instruction set. "Programmer's Reference Manual for Intel 80386" [8] (1986) already contains 421 pages. Finally, the current edition of "Intel 64 and IA-32 Architectures Software Developers Manual" (2018) [9] has 2214 pages describing instructions.

Even Intel 8086 was a processor with CISC design and with a set of instructions of variable length (from 1 byte and up to 4 bytes) and with variable execution time. With time going, IA-32 became more and more CISC-like. Currently, it supports a set of more than 200 instructions with lengths varying starting from 1 byte and ending by 15 bytes.

IA-32 supports multiple addressing modes and complex memory access mechanisms. Most of the instructions in that ISA can reference memory directly. In contrast to IA-32, RISC ISAs commonly rely on an especial pair of instructions dedicated to data exchange between memory and registers while rest instructions operate exclusively on registers. Availability of two memory addressing modes and two functioning modes bring additional complexity to IA-32 architecture.

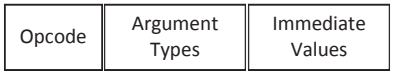


Fig. 1: A general structure of binary instruction format of IA-32 ISA.

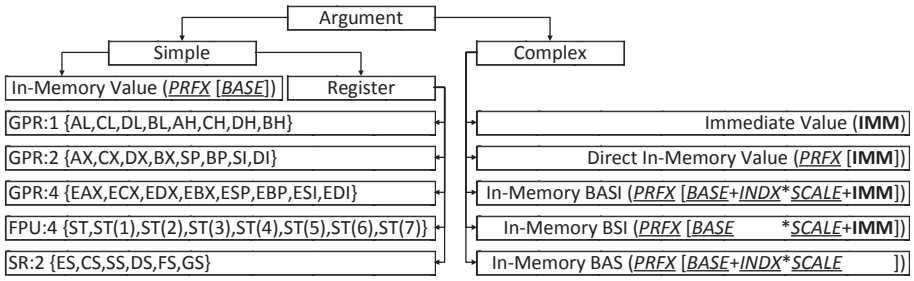


Fig. 2: Instruction argument types supported by the IA-32 instruction set.

2.1 Instructions on IA-32

Instructions in IA-32 have the following general structure depicted in Figure 1. As can be seen, every IA-32 instruction consists of three components: opcode, argument types block and immediate values block, where only the opcode is mandatory part while other parts are optional. Opcode bytes not only define instruction behavior but also guides decoder about rest of the instruction bytes.

IA-32 instruction can have from 0 up to 3 either primitive or composite arguments, as shown on Figure 2. The second ones either comes with immediate value or use multiple registers. *BASE INDX* are one of the registers from the set *GPR:4*. *PRFX* can accompany any memory-referencing argument to specify the size of the referenced value explicitly. The list of prefixes includes byte, word, dword, fword, qword, and tword (1,2,4,6,8 and 10 bytes respectively). *SCALE* can take only values 1, 2, 4, and 8. *IMM* denotes an immediate value.

From a semantics viewpoint, every instruction in IA-32 consists of a command, three arguments and two immediate values associated with them. First one is the only mandatory component. All other instruction parts are optional and can be void. At the same time, only immediate values explicitly referenced by arguments become meaningful.

2.2 Classification of IA-32 instructions

In contrast to RISC systems, IA-32 does not have uniform instruction encoding. Besides that, there are few families of instructions. Each instruction from the same family follows the same encoding rules. There are four general families of instructions which are distributed over IA-32 decoding root map as depicted in Figure 3.

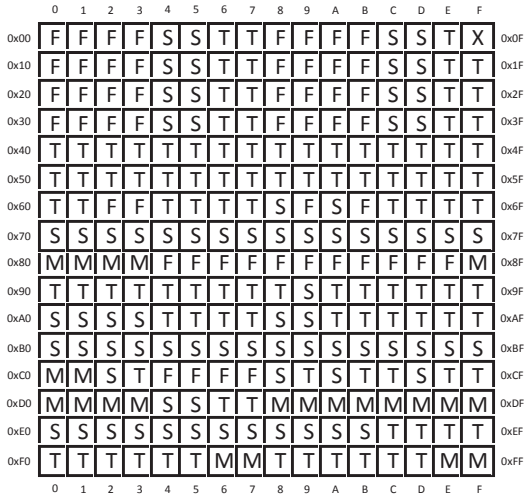


Fig. 3: Map of IA-32 decoding tree roots.

Tiny instructions. Tiny instructions represent a set of instructions for which one or two bytes define entire semantics of instruction. Almost all commands (instructions which do not have operands) (*std*), frequently used predicates (instructions with only one operand) with in-register arguments (*inc*) and even some operations (instructions with two operands) with in-register only arguments (for example *xchg eax, ecx*) fall into that family.

Snap instructions. Snap instructions represent a family of instructions which consist of one command byte and one immediate value following it. There are two subclasses of snap instructions: predicates with an immediate value (*push imm*), and operations with one fixed in-register argument and an immediate value as a second argument (*add eax, imm*). The first byte of the instruction defines its entire semantics, while the immediate value defines its operands.

Instructions with fixed and mixed commands. There is the only difference between these families. In instructions with fixed command, the first byte of instruction explicitly defines command encoded. Instructions with mixed commands use at least two bytes for command encoding. Both families consist of encoding trees each leaf of which contains instructions with particular semantics and encoding.

2.3 Encoding trees and instructions types.

IA-32 includes three types of decoding trees:

- Tree of completely manageable register-based operations.
- Tree of semi-manageable register-based operations.
- Tree of operations with an immediate argument.

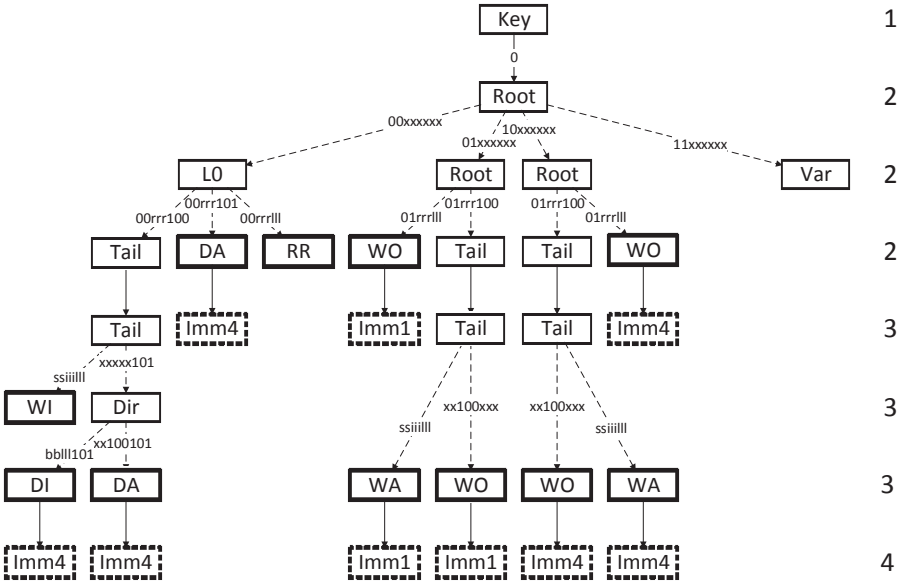


Fig. 4: Structure of IA-32 encoding tree.

Each tree contains four leaves on the first level, where first three leaves create three stable triplets that define a type of encoding subtree. The fourth leaf is highly-variable and differs between different decoding trees. Nevertheless, all decoding trees follow the same structure depicted in Figure 4. All these complexities introduced by encoding trees are a direct consequence of support of multiple addressing modes listed above.

3 General architecture of EIDIA decoder

The best approach to the feeding of IA-32 instruction decoder is feeding in byte-by-byte fashion. In that case, front-end receives control over decoder and on its input stream of code bytes after processing of each byte of code. Thus, front-end forms the input byte stream, while there is no intermediate buffering. Moreover, this feeding scheme completely releases decoder from feeding management and control tasks. At the same time, the byte-by-byte feeding scheme implies conditional generation of output instruction description. Figure 5 presents the scheme of the interaction of front-end with EIDIA decoder. As can be seen on the figure, the decoder can be in three states: decoding complete, decoding incomplete, and undefined instruction found. When front-end founds decoder in "Decoding incomplete" state, it routes self through a fast path to next feeding round. Otherwise, it captures and processes decoded instruction (in case of "decoding complete" state) or handles the exceptional situation (in case of "undefined instruction" state).

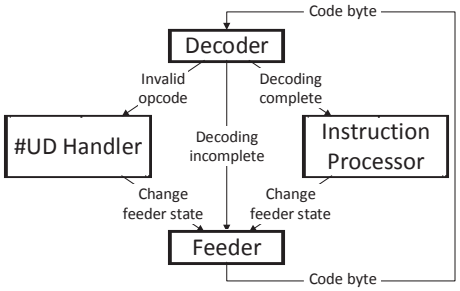


Fig. 5: Scheme of interaction with EIDIA decoder.

The architecture of interfacing with decoder depicted in Figure 6 also reflects the fact that it can be beneficial to consider decoder as a state machine in general or as Mealy machine automata in particular and design decoder in the appropriate way. Automata's input is bytes of code. Internal states of decoding can be represented as a graph of handlers, while the internal state of decoder, in that case, will be represented by function pointer defining the current state of decoding. The decoder has an initial state which represents start point in instruction decoding, but it has no distinct finish state. Each time when decoder either have instruction successfully decoded or have undefined opcode detected it reports that decoding was finished and switches self to initial state. Therefore, each state from which decoder can directly transit into the initial state can be considered finish state.

Mealy machine basis of decoder allows applying extensive table-based dispatching of decoding. Multiple tables are in use. Routing tables switch decoder onto appropriate handler depending on code byte at the input. Additional semantic data table contains a generalized description of the instruction set. Different parts of decoder use this table which increases the uniformity of decoder code, thus, improving the efficiency of CPU cache usage. At the same time, during each decoding step EIDIA accumulates information about instruction decoded, as well as, information which will guide decision making during next decoding steps. Thus, EIDIA does not use instruction bytes as a path to the complete instruction description, but assembles the description in a step-by-step way. Furthermore, EIDIA reconfigures itself during each decoding step.

Following the state machine architecture with byte-by-byte feeding eliminates all external dependencies from EIDIA. For example, EIDIA does not dynamically allocate or manipulate memory and does not use C standard library at all. Furthermore, EIDIA does not involved into the instruction byte stream management which eliminates frequently redundant preparations and checks of the input byte stream.

4 Output interface of EIDIA

The output interface of decoder should be convenient for use and completely cover general semantic of instruction. EIDIA returns instruction description represented in the form of a pointer to the next data structure:

```
struct Instruction {
  uint32_t command;
  uint32_t args[3];
  uint32_t imms[2];
};
```

The output of the decoder has a size of 24 bytes and can incur significant overhead on memory copying during transfer from the decoder to its front-end. Thus, the efficient decoder should have an internal buffer which it uses for instruction construction during decoding. Decoder exports interface for accessing that buffer to the front-end. Therefore, when the back-end receives status "Decoding complete" that status serves it as a signal that front-end can safely capture instruction from the internal buffer using the provided interface. Furthermore, front-end becomes able to perform access only those components of instruction description which contain actual data. For example, if decoded instruction is a command (have no arguments), then backend can read command opcode, using it determine that there are no arguments and finish work with decoder buffer, hence performing access only to 4 bytes from 24 available bytes of the buffer.

IA-32 has a subset of instructions with a fixed command and which have encoding trees with reverse order of instruction arguments. They have the same encoding as regular instruction. Encoding tree itself and hence first byte of instruction specifies the reverse ordering of arguments. Interface for access to internal buffer allows simplifying decoder internals because it preserves uniformity of decoding algorithms. Such an especial interface can reverse arguments order for backend at access time by logical mapping of the external view of the buffer fields to respective internal implementation.

Especial interface to decoder buffer implements lazy output reset. Lazy reset moves the burden of internal buffer cleanup from the stage of decoding to the results fetching stage, which leads to performance penalty reduction. Decoder front-end can cleanup only those fields of the internal buffer which were set by decoder during assembling of instruction. At the same time, lazy reset can eliminate redundant resetting of those fields which were not modified by the decoder.

Finally, an especial interface provides an opportunity for extensive precaching of ready-to-use generalized instruction descriptions. In case of decoding of tiny instructions, the decoder can point output interface onto appropriate already ready-to-use instruction description instead of filling of the default output buffer.

	Platform A	Platform B	Platform C
CPU	Intel Core i7-4600U	AMD Phenom FX-8350	Intel Core i7-7500U
Architecture	Haswell	Bulldozer	Kaby Lake
Codename	Haswell-ULT	Piledriver	Kaby Lake-U
Frequency	2100 MHz	4000 MHz	2700 MHz
L1D cache	2 x 32KB	8 x 16KB	2 x 32KB
L1I cache	2 x 32KB	4 x 64KB	2 x 32KB
L2 cache	2 x 256KB	4 x 2MB	2 x 256KB
L3 cache	4MB	8MB	4MB

Table 1: Hardware used for performance evaluation

5 Evaluation

Table 1 presents the specification of CPUs of computer systems which we have used for evaluation.

We have measured throughput of EIDIA in two scenarios. The first scenario is pure binary instruction stream decoding. Results achieved for this scenario show throughput in raw instructions decoding. In the second scenario, we have measured throughput of the decoder which has backend attached. In the role of the backend, we have used simple disassembler application that was designed and implemented from scratch.

Finally, we have used two types of workload. Specially generated file containing all variants of IA-32 instructions (2,1152MB, 445215 instructions) represents synthetic workload. In the role of real-world workload, we have used code sections extracted from Linux kernel file of version *3.13.0-37-generic* (6.7546MB, 2141376 instructions).

To proof performance benefits of EIDIA, we have compared it with two IA-32 instruction decoders: *Udis86* [6] and *Intel XED* [3]. Both decoders have disassembler capabilities.

It would be interesting to compare EIDIA with decoders used in emulators and virtual machines. However, such decoders are an integral part of VM execution engines, and their extraction is a nontrivial task. Furthermore, they do not have disassembler backends which prevent macrobenchmarking.

The results of measurements are summarized in the Table 2. The numbers in that table show the speedup in processing time achieved by EIDIA in comparison to respective decoder specified in the column header. As can be seen, EIDIA is from 21.09 up to 51.11 times more performant than UDis86 in pure instruction decoding and from 7.54 up to 13.48 times more performant in disassembling tasks. What is more important, the proposed solution provides throughput from 3.48 to 4.45 times better than Intel XED in instruction decoding tasks and from 5.8 to 13.79 times better in disassembling tasks. EIDIA has demonstrated at least 3.48 times better performance in all conducted experiments, and at the same time stays agnostic to the underlying hardware platform and provides clear isolation of decoding from front-end logic.

Task	Workload	Platform	Udis86	Intel XED
Decoding	Synthetic	A	42,03	4,19
Decoding	Synthetic	B	31,13	3,58
Decoding	Synthetic	C	51,11	4,45
Decoding	Real-World	A	23,08	3,48
Decoding	Real-World	B	21,09	3,83
Decoding	Real-World	C	24,64	3,64
Disassembling	Synthetic	A	12,26	11,15
Disassembling	Synthetic	B	10,95	13,79
Disassembling	Synthetic	C	13,48	10,67
Disassembling	Real-World	A	9,16	6,18
Disassembling	Real-World	B	7,54	7,51
Disassembling	Real-World	C	9,15	5,80

Table 2: Speedup of the EIDIA decoder in instruction decoding comparing to other instruction decoders for IA-32

6 Conclusion

In this paper, we have explored techniques of efficient instruction decoding for IA-32. Our instructions decoder – EIDIA demonstrates that high-throughput general purpose decoder based on Mealy machine with byte-by-byte feeding delivers high performance. Our instruction decoder exploits several performance-oriented features including extensive precaching of decoded instructions; multi-level table-guided dispatching; lazy reset of output and compact description of instruction semantics. This features in application to the design of state machine make decoder performant while preserving its general purpose nature. We have compared EIDIA with analogs of industrial quality. The measurements have shown that EIDIA has from 3.5 up to 4.5 times higher throughput than Intel XED in case of pure decoding, and from 5.8 up to 13.8 times better in case of disassembling. At the same time, EIDIA, in contrast to Intel XED, is agnostic to host CPU.

References

1. bochs: The Open Source IA-32 Emulation Project (Home Page). http://bochs.sourceforge.net/, 2017.
2. gem5. http://gem5.org/Main_Page, 2017.
3. Intel XED. https://intelxed.github.io/, 2017.
4. QEMU. https://www.qemu.org/, 2017.
5. GitHub - dyninst/dyninst: DyninstAPI: Tools for binary instrumentation, analysis, and modification. https://github.com/dyninst/dyninst, 2018.
6. Udis86 Disassembler Library for x86 / x86-64. http://udis86.sourceforge.net/, 2018.

7. Intel Corporation. *The 8086 Family Users Manual*. No. 9800722-03. October 1979.
8. Intel Corporation. *Intel 80386 Programmer's Reference Manual*. No. 230985. 1986.
9. Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer's Manual: Instruction Set Reference* . No. 325383-066US. March 2018.
10. Reshadi, M., Dutt, N. D., and Mishra, P. A retargetable framework for instruction-set architecture simulation. *ACM Trans. Embedded Comput. Syst. 5* (2006), 431–452.

Prediction of RNA Secondary Structure Based on Optimization in The Space of Its Descriptors by The Simulated Annealing Algorithm

Nikolay Kobalo[1], Alexander Kulikov[3] and Igor Titov[3]

[1] Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Russia
rerf2010rerf@yandex.ru
[2] Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Russia
kulikovai12@gmail.com
[3] Institute of Cytology and Genetics SB RAS, Russia
titov@bionet.nsc.ru

Abstract. The proportion of genome coding proteins is only a small part of a whole genome (for example, about 5% in human's genome). Among other things the remaining part contains regulatory RNAs whose function depends on their three-dimensional structure. Secondary structure is the first level of RNA structure description (three-dimensional structure is approximated by secondary structure).

Therefore the problem of determining the common secondary structure of isofunctional RNA sequences (i.e., a set having similar functionality) is an important and longstanding problem of bioinformatics. In this paper we present the program which builds the secondary structure model for a such set of non-homologous RNA sequences.

Secondary structure is described by directed acyclic graph i.e. multitree. The problem of determining the model of secondary structure is reduced to the discrete optimization task in the space of structure multitrees. The optimizable function depends on the energy of the referenced sequences being folded into this structure.

The optimization task is solved by simulated annealing algorithm. We developed the program for building a common secondary structure model of RNA and compared it with the existing solutions on the set of mobile group II introns.

Keywords: RNA, secondary structure prediction, mobile group II intron, optimization, simulated annealing, software.

1 Introduction

The secondary structure of RNA describes the interactions between the nucleotide bases of RNA molecule. Due to these interactions, the RNA can fold into a complex spatial configuration. The secondary structure is the first level of description of this configuration.

The RNA sequence is modeled by a symbol sequence, where each of the 4 nucleotides represented in the RNA has its own symbol: A (Adenine), C (Cytosine), G (Guanine) and U (Uracil). The secondary structure of RNA is formed due to the ability of nucleotides to chemically bind, forming pairs: A-U, G-U or G-C.

In Fig. 1 shows an example of a tRNA secondary structure and shows the elements of which it is composed. This diagram shows how the one-dimensional nucleotide sequence folds into a two-dimensional structure, due to the formation of binds between individual nucleotides. As can be seen, the secondary structure consists of two main types of elements - the stems formed by paired nucleotides and the loops - free sequences of unrelated nucleotides.

An important task of bioinformatics is the problem of predicting the secondary structure of RNA by its nucleotide sequence. The same sequence can be folded into a large number of different secondary structures, therefore, in order to correctly predict the real structure, it is necessary to draw additional assumptions. One of these assumptions is that a sequence that is realized in nature must possess the minimum possible thermodynamic energy. In addition, RNA with similar functionality in an organism will likely have a similar secondary structure, so for such sequences a general structure that satisfies the principle of minimum energy can be found.

There are many algorithms and software implementations designed to predict the secondary structure of single sequences (RNAFold, MFold, SARNA-Predict), as well as the common structure of a sequence group (RNAStructure Multialign, RNAStructure Multialign, PFold). However, currently existing methods make it possible to effectively build models of secondary structures only for small sets of sequences. In addition, they do not work well with sets of low-homologous sequences and do not allow to take into account *a priori* information about the structure of sequences.

This paper presents a new method for constructing a secondary structure model for a set of sequences. This method is based on reducing the problem of constructing a model to the problem of discrete optimization in the space of all possible models, and the optimized parameter is the energy of the resulting structure [1].

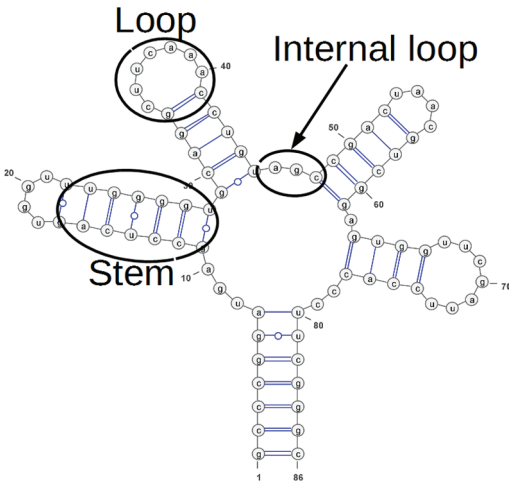


Fig. 1. An example of the secondary structure of RNA

2 Basic requirements for the secondary structure prediction method

The following requirements are imposed on our method.

- Ability to build models for sets of sequences of any size. At the same time, it should detect and correctly handle the situation when the set is a mixture of sequences that actually have different secondary structure.
- Ability to build models for sequences that are not as similar as possible.
- Sometimes information about the secondary structure of sequences of a certain type (for example, introns or tRNA) is known in advance. In this case, the task is reduced to the construction of a refined secondary structure that satisfies the given general constraints. Therefore, the method should allow to set this *a priori* information and take it into account when building a model.
- After setting the initial parameters and *a priori* information about the structure, the system should work in automatic mode and not require manual intervention.
- The system must provide a practically acceptable speed of building a model.

3 Review of existing software for predicting the secondary structure of RNA

There are many algorithms and their implementations for predicting the secondary structure of RNA sequences, for example RNAFold, mfold, CentroidFold, RNAStructure Multialign, RNAStructure Multialign, PFold, SARNA-Predict.

Let us consider in more detail those methods that satisfy the requirements for the algorithm formulated above.

- RNAStructure Multialign - predicts the secondary structure of a set of three or more RNA sequences using the minimum energy estimate. [2]
- RNAStructure TurboFold - predicts the secondary structure of two or more sequences. It generates pairwise alignments for the set using a hidden markov model, which supplies extrinsic information to one of three selectable folding modes. [3]
- PFold - this algorithm allows to specify some *a priori* information about the secondary structure, such as the exact position of the symbols that should be paired in the resulting structure or, in contrast, free [4].

4 Materials and Methods

4.1 Programs and Test Data

- Our method uses the RScan program [5] to determine the correspondence of the constructed model of the secondary structure to specific sequences and calculate its energy. The energy is measured by RScan in ES units used by RScan, which is energy in kcal/mol, multiplied by 10 and taken with the opposite sign.
- To test the program, a set of 40 sequences was used, each 72 symbols in length, with the same secondary structure shown in Fig. 4. These sequences were taken from rfam [6]. For sequences from this set, the value of the optimal secondary structure energy was calculated by the RNAFold program, which builds an optimal secondary structure for single sequences [7].
- Sequences from the family with a known secondary structure were used - the sequences of the first domain of the following mobile introns of group II [8,9] were used: Pylaiella littoralis cox1.I3 and 8 other introns, with the first domain similar to it in its secondary structure: Thalassiosira pseudooana cox1.I2, Allomyces macrogynus cox1.I3, Podaspora anserina cox1.I1, Podaspora anserina cox1.I4, Podaspora comata cox1.I1, Kluyveromyces lactis cox1.I1, Saccharomyces cerevisiae cox1.I2 and Schizosaccharomyces pombe cox1.I1. The average length of these sequences is 405 nucleotides.
- To compare programs, a set of 10 tRNA sequences from rfam tRNA-Sec RF01852 [10] were also used. The average length of the sequences of this set is 89 symbols.

4.2 Data Representation

The model of the secondary structure of RNA is represented as an oriented acyclic graph — a multi-tree. In this case, the stems are represented by the edges of a multi-tree, and the loops – by the vertices.

Each element of the tree has a set of attributes that define restrictions on the elements of the secondary structure. The following attributes are supported:

- Permitted length range of elements (loops and stems) in symbols.
- The sequence of nucleotides, which must necessarily be present on this element and its position relative to the beginning of the element.

For example, in Fig. 2b shows an example of a secondary structure model, and Fig. 2a - its representation in the form of a multi-tree.

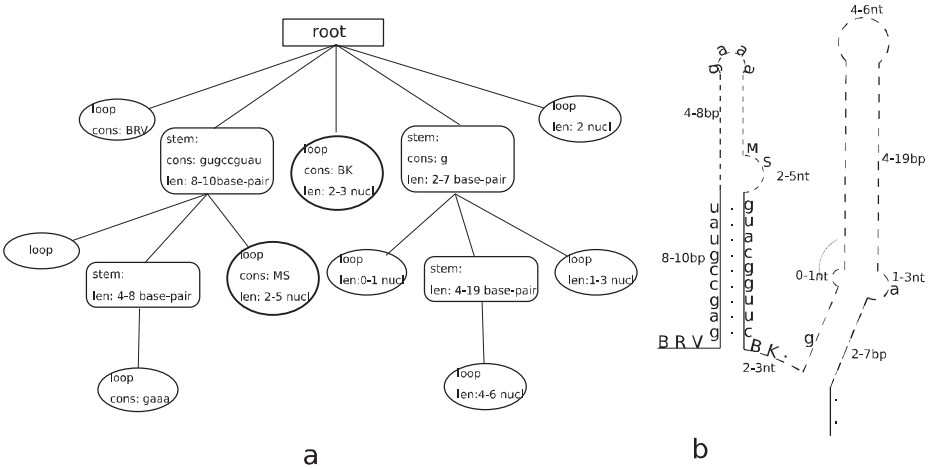


Fig. 2. a: Multi-tree modeling the secondary structure of RNA; b: The corresponding secondary structure of RNA. The lengths of the stems are expressed in bp - the number of paired symbols forming the stem, and the length of the loops in nt - is the length of the sequence forming the loop in the symbols. The symbols B, R, V, K, M mean that at this place in the sequence there can be not one specific symbol, but one of the set of symbols: B = U or G or C, R = A or G, V = A or G or C, K = U or G, M = A or C

5 Method Description

5.1 Reducing to Discrete Optimization Task

The problem of building a model of the secondary structure of a set of sequences is reduced to the discrete optimization task as follows:

Let be:

- T– is the set of all admissible multi-trees representing the secondary structure.
- $S = \{s \mid s \in n^*, n \in \{a, u, g, c\}\}$ - is the finite set of words in the alphabet a, u, g, c, representing the set of sequences for which the secondary structure model is built.
- $R(t,s)$: $T \times S \to \mathbb{R}$ - is a function that calculates for a given multi-tree and sequence the value of the energy of a given sequence, folded into a given structure. $R(t,s) = \infty$, if the sequence s cannot be folded into the structure t.
- $C(t,S)$ - is the number of sequences $s \in S$, such that $R(t,s) < \infty$.
- $E(t,S)$ – is the average energy for all sequences $s \in S$, such that $R(t,s) < \infty$.
- $T(t,S)$ – is the average computation time for $R(t,S)$ in milliseconds.

Then the expression $F(x) = -k_1 E(t,S) + k_2 T(t,S) - k_3 C(t,S)$ defines the objective function for the problem of discrete optimization in the multi-tree space T.

Coefficients k_1, k_2, k_3 are selected in each specific case manually and set when the program is started.

In this expression, the first term takes into account the energy of the secondary structure, which is a measure of its stability and should be minimized. The second term allows to take into account the calculation time of the objective function using the RScan program. The inclusion of this term in the objective function is important from a practical point of view, because it allows to speed up the calculation. The third term of the expression allows to build a model for as many sequences as possible from the original set. At the same time, it allows the algorithm to correctly handle the situation when the set of sequences is an actual mixture of sets with different secondary structure.

5.2 Solution of the optimization task

To solve the optimization task described above, an annealing simulation algorithm was applied.

```
T = T0
While cost function changes
  Temp = Kt*Temp
  Tnew = modifyTree(T)
  If P(T, Tnew, Temp)
    T = Tnew
Result = T
```

Here:

- $0 < K_t < 1$ - coefficient of temperature change
- P(T, T_{new}, Temp) - function that takes the true value with the following probability:

$$f(x) = \begin{cases} 1, & R(T,S) < R(T_{new},S) \\ \exp(-\dfrac{R(T,S) - R(T_{new},S)}{Temp}), & R(T,S) \geq R(T_{new},S) \end{cases}$$

To start the computation, you must specify some initial model of the secondary structure T_0. The `modifyTree` is a function that applies one or more mutation operators to a multi-tree:

- Changing the value of a numeric attribute of a multi-tree element (for example, the range of lengths or the position of the consensus sequence, if specified).
- Adding a leaf to the tree, or deleting an existing one.
- Adding a vertex to an arbitrary multi-tree location, or deleting an existing one.

All the described operators select a part of the tree for modification at random. The last two operators correspond to the addition or removal from the tree of a random element of the secondary structure - a stem or loop, as shown in Fig.3a and Fig. 3b.

The described algorithm was implemented in java and is available on github [11]. To calculate the value of the function $R(t, s)$, the RScan program is used.

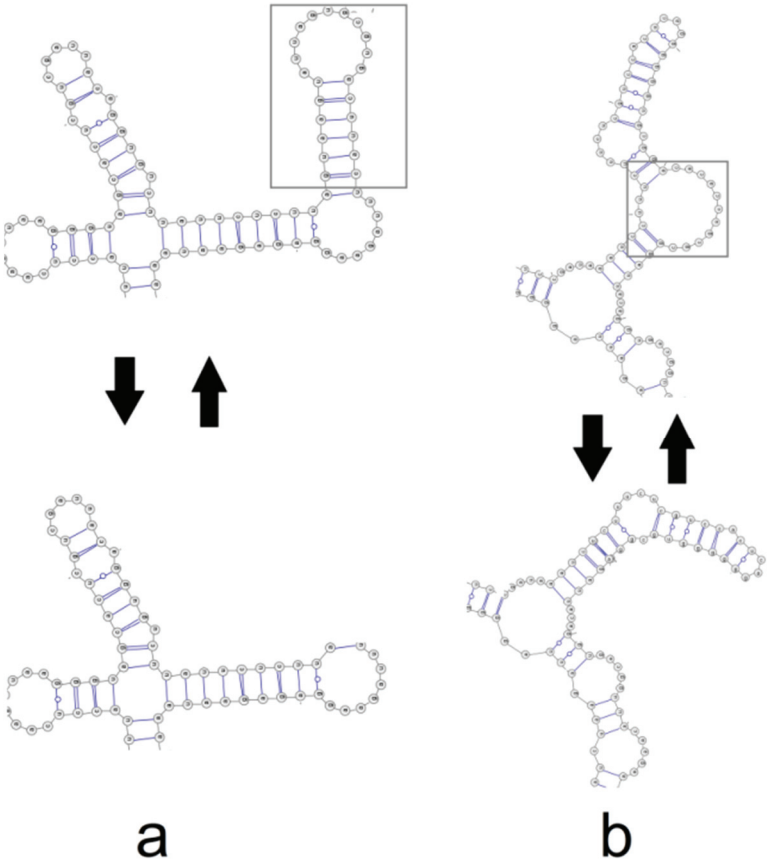


Fig. 3. a: Modification of the secondary structure by adding or removing a multi-tree leaf:; b: Modification of the secondary structure by adding or removing the inner vertex of a multi-tree.

6 Testing

6.1 Evaluation of the accuracy of the solution

To verify the accuracy of solving the optimization task found by the implemented algorithm, a set of 40 sequences with the same optimal secondary structure was formed and the implemented program was launched on it. The test sequences were taken from the rfam tRNA RF00005 family. Their reference secondary structure was constructed using the RNAFold program, and sequences with the same secondary structure were selected.

The following parameters were used: $k_1 = 10, k_2 = 1, k_3 = 10$.

Table 1 shows the optimal and program-determined values of each term of the objective function.

Table 1. Comparison of optimal and predicted secondary structure models

Structure	Time of computation, ms	Optimal energy, $ES = -10 \cdot kcal/mol$	Number of sequences	Cost function
Optimal	40	144	40	-1800
Predicted	169	126	40	-1491

The sensitivity and F-measure were also calculated [12]. Sensitivity and f-measure measures the quality of the binary classification algorithm.

Let the classification algorithm answer the question whether a given object belongs to a certain class or not. Then sensitivity expresses the ratio of number of objects correctly assigned by the algorithm to this class to the number of objects that actually belong to this class.

Specificity is the ratio of the number of objects not related by an algorithm to a class of objects to the number of objects, which are really not related to this class.

Then the F-measure is calculated as follows:

$$F - measure = 2\frac{sensetivity * specificity}{sensetivity + specificity}$$

In our case, the objects that need to be classified were the symbols pairs in the sequence, which in the secondary structure must be paired.

We got the following result:

- Sensetivity = 0.9
- F-Measure = 0.95

In Fig. 4b shows the optimal secondary structure of the considered set of sequences, and Fig. 4a - predicted by the program.

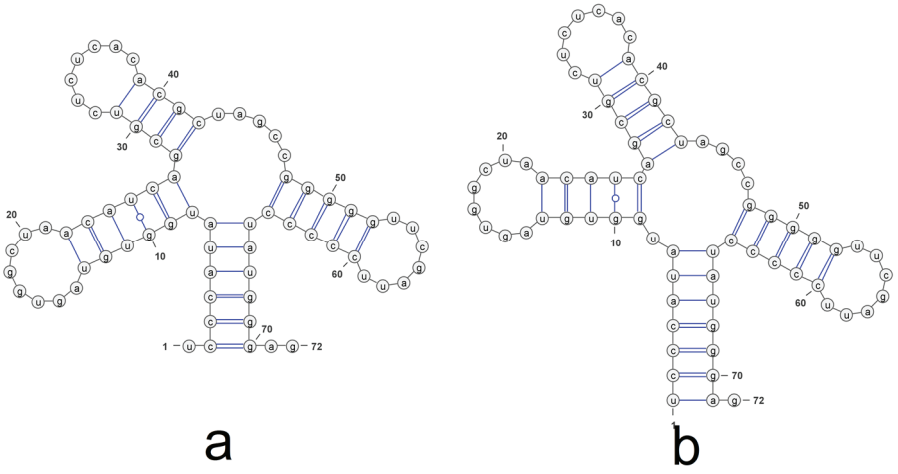


Fig. 4. a: Predicted secondary structure for test set; b: Optimal secondary structure of the test set.

6.2 Comparison of programs

The program was also tested on a sample of the rfam family tRNA-Sec sequences. For each predicted structure, the number of erroneously predicted paired bases was calculated. Then, sensitivity and F-measure were calculated for all sequences. The results obtained for all the programs compared are shown in Table 2.

Table 2. Comparison of programs on a sample of transport RNA sequences

Program	Sensitivity	F-measure	Time of computation
Simulated Annealing	0.79	0.8	13 minutes
PFold	0.42	0.5	2 seconds.
RNAStructure Multialign	0.8	0.82	7 minutes
RNAStructure TurboFold	0.79	0.82	40 seconds.

Also, the programs were launched on a set of 9 group II introns. We took those introns, the secondary structure of which best corresponds to the generalized group II intron secondary structure [13].

Table 3 shows the program comparison

Table 3. Comparison of programs on a sample of group II introns

Program	Sensitivity	F-measure	Time of computation
Simulated	0.51	0.53	~112 hours.

Annealing			
PFold	0.06	0.07	6 seconds.
RNAStructure Multialign	0.38	0.4	~21 hours.
RNAStructure TurboFold	0.46	0.47	320 seconds.

As one can see from the Tables 2-3, on the set of small sequences of tRNA with an average length of 89 nucleotides, the proposed method gives the results comparable with other existing methods. However, for large sequences of group II introns with an average length of 405 nucleotides and complex secondary structure, our method gives more accurate results.

Acknowledgements

The work of I.T. was supported by the Federal Agency of Scientific Organizations (project #0324-2019-0040).

References

1. Skiena, Steven. The Algorithm Design Manual (2nd ed.). Springer Science+Business Media (2010)
2. Reuter, Jessica & H. Mathews, David. (2009). RNAstructure: Software for RNA Secondary Structure Prediction and Analysis. Journal of biomolecular Structure & Dynamics. 26. 831-832.
3. Harmanci, A.O., Sharma, G., and Mathews, D.H.: TurboFold: Iterative Probabilistic Estimation of Secondary Structures for Multiple RNA Sequences. BMC Bioinformatics, 12:108. (2011). doi: 10.1186/1471-2105-12-108
4. Z. Sukosd, B. Knudsen, J. Kjems, C. N. S. Pedersen.: PPfold 3.0: PPfold 3.0: Fast RNA secondary structure prediction using phylogeny and auxiliary data. Bioinformatics 28(16), 2012. doi: 10.1093/bioinformatics/bts488
5. http://www.softberry.com/freedownloadhelp/rna/rscan/rscan.all.html
6. Rfam family of tRNA http://rfam.xfam.org/family/RF00005
7. Jaeger, J. A., Turner, D. H. and Zuker, M.: Improved predictions of secondary structures for RNA. Proc. Natl Acad. Sci. USA (1989), 86, 7706-7710
8. Manuel A. Candales, Adrian Duong, Keyar S. Hood, Tony Li, Ryan A. E. Neufeld, Runda Sun, Bonnie A. McNeil, Li Wu, Ashley M. Jarding, and Steven Zimmerly.: Database for bacterial group II introns. Nucleic Acids Research (2012) 187-190. doi: 10.1093/nar/gkr1043
9. Jean-Marc Fontaine, Didier Goux, Bernard Kloareg, Susan Loiseaux-de Goer.: The Reverse-Transcriptase-Like Proteins Encoded by Group II Introns in the Mitochondrial Genome of the Brown Alga Pylaiella littoralis Belong to Two Different Lineages Which Apparently Coevolved with the Group II Ribosyme Lineages. J Mol Evol (1997) 44:33–42. doi:10.1007/PL00006119
10. Rfam family of selenocysteine transfer RNA http://rfam.xfam.org/family/RF01852

11. https://github.com/rerf2010rerf/RNAStructBuilder
12. Powers, David M W (2011). "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation" (PDF). Journal of Machine Learning Technologies. 2 (1): 37–63
13. Steven Zimmerly and Cameron Semper.: Evolution of group II introns. Zimmerly and Semper Mobile DNA (2015) 6:7. doi: 10.1186/s13100-015-0037-5

Towards Automatic Deductive Verification of C Programs Over Linear Arrays*

Dmitry Kondratyev[0000−0002−9387−6735], Ilya Maryasov[0000−0002−2497−6484], and Valery Nepomiaschy[0000−0003−1364−5281]

A. P. Ershov Institute of Informatics Systems,
Siberian Branch of the Russian Academy of Sciences
6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia
apple-66@mail.ru, {ivm, vnep}@iis.nsk.su
http://www.iis.nsk.su

Abstract. The generation and proving of verification conditions, which correspond to loops, may cause difficulties during deductive verification because the construction of required invariants is a challenge, especially for nested loops. The methods of invariant synthesis are often heuristic ones. Another way is the symbolic method of loop invariant elimination. Its idea is to represent a loop body in a form of special replacement operation under certain constraints. This operation expresses loop effect with possible `break` statement in a symbolic form and allows introducing an inference rule, which uses no invariants in axiomatic semantics. This work represents the further development of this method. The inner loops are interesting because of the higher nesting level, the more complicated loop invariant. A good example for this case to verify is a class of linear array sorting programs, which iteratively increase the sorted part. In this paper, we consider the insertion sort program. A special algorithm was developed and implemented to prove verification conditions automatically in ACL2. It generates automatically auxiliary lemmas, which allow to prove obtained verification conditions in ACL2 successfully in automatic mode.

Keywords: C-light · Loop invariants · Mixed axiomatic semantics · Definite iteration · Arrays · Sorting · ACL2 · Verification · Hoare logic.

1 Introduction

C program verification is an urgent problem today. Some projects (e.g. [2, 4]) suggests different solutions. But none of them contains any methods for automatic verification of loop-containing programs without invariants. As it is known, in order to verify loops we need invariants whose construction is a challenge. Therefore, the user has to provide these invariants. For many cases, it is a difficult task.

* This research is partially supported by RFBR grant 17-01-00789.

Tuerk [16] suggested to use pre- and post-conditions for while-loops, but the user still has to construct them himself. Li et al. [10] developed a learning algorithm of loop invariants generation, but their method does not support array operations and the `break` statement in the loop body. Galeotti et al. [5] improved a well-known method of post-condition mutation by a combination of test case generation and dynamic invariant detection. However, this approach failed to infer full invariant for sorting programs. Srivastava et al. [15] proposed a method, which is based on user-provided invariant templates. This method also is not able to perform a full verification of sorting programs. Kovács [9] developed the method of the automatic invariant generation for the P-solvable loops, where right operands of assignment statements in the loop body must have a form of polynomial expression and the `break` statement is not considered.

We consider loops with certain restrictions [14]. We extend our mixed axiomatic semantics of the C-light language [1] with a new rule for verification of such loops, based on the replacement operation [14]. The special verification conditions are generated with the help of this rule.

In our previous paper [8], we considered the strategy of automatic proving of verification conditions. It can be applied, if the program specification describes whether the `break` statement occurred or not. The new goal is the verification of the insertion sort program, which has different specification.

Sorting programs are programs over changeable arrays with loop exit. Their specifications contain functions with concatenation property. These programs can contain downward loops and can use the value of loop counter after iterations are finished. Thus, we had to change our algorithm of replacement operation generation. Also, we overcame the difficulties in proving verification conditions by developing new strategies of proof. This paper describes the solution of these problems.

2 Preliminary Concepts

We develop a two-level system of deductive verification of the C-light programs [13]. The C-light language is a powerful subset of C language. To prove obtained in our system verification conditions, we use the theorem prover ACL2 [6].

The input language of ACL2 is an applicative dialect of Common Lisp language, which supports only functional paradigm and does not support imperative one.

Since Common Lisp language focuses on list processing, arrays of the C-light language we simulate by lists. Consider list operations in ACL2. If *expr* is an expression of ACL2 language, then (*update-nth i expr l*) is a new list, which coincides with a list l except for i-th element, whose value is *expr*. The function *len* returns the length of a list.

To verify programs without invariants, we implemented the method of loop invariants elimination for definite iteration [14] in our system. Our previous works [11, 12] dealt with definite iteration over unchangeable data structures

with / without loop exit. In this paper, we moved to changeable data structures with possible `break` statement in the loop body.

Consider the statement `for x in S do v := body(v, x) end`, where S is a structure, x is the variable of the type "an element S", v is a vector of loop variables, which does not contain x and *body* represents the loop body computation, which does not modify x and which terminates for each $x \in S$. The structure S can be modified as described below. The loop body can contain only the assignment statements, the `if` statements, possibly nested, and the `break` statements. Such `for` statement is named a definite iteration. Let v_0 be the vector of values of variables from v just before the loop. To express the effect of the iteration let us define a replacement operation $rep(v, S, body, n)$, where $rep(v, S, body, 0) = v_0$, $rep(v, S, body, i) = body(rep(v, S, body, i-1), s_i)$ for all $i = 1, 2, \ldots, n$.

A number of theorems, which express important properties of the replacement operation, were proved in [14].

The inference rule for definite iterations has the form:

$$\frac{\{P\}\ \mathbf{A;}\ \{Q(v \leftarrow rep(v, S, body, n))\}}{\{P\}\ \mathbf{A;\ for\ x\ in\ S\ do\ v := body(v,\ x)\ end}\ \{Q\}}$$

Here A are program statements before the loop. We find the weakest precondition applying the mixed axiomatic semantics [1] of the C-light language.

3 Generation of Replacement Operation

In this paper, we extend the class of definite iterations by downward iteration

```
for (i = n - 1; i >= 0; i--) v := body(v, i) end,
```

where n is the number of elements of array.

The definition of *rep* is generated by a special translator [8]. The statements of the loop body are translated to the constructions of the ACL2 language. The fields of the structure of the type $frame$ correspond to the variables of v and the function $frame$-*init* creates the object of the type $frame$ with given field values.

For downward iteration, the generator of *rep* was modified. Firstly, it has to generate not only the structure $frame$ but the structure *envir*. It stores the values of variables, which are used but are not modified inside the loop. Moreover, the generator makes a definition of the function *envir-init*, which creates an object of the type *envir* with given field values. The structure *envir* has also a dedicated field *upper-bound*, whose value is an inaccessible upper bound of the loop counter, which is equal to $(n-1)+1$.

Secondly, in the case of downward iteration, the value of loop counter is not equal to the number of iteration. In order to distinguish from loop counter i, the first argument of *rep* is called *iteration*. Therefore, the generator has to include not only the elements of the vector v, but also the loop counter to the fields of

the structure $frame$. This allows using the value of the loop counter after loop execution.

One of the restrictions of the symbolic method [14] is that the loop counter is not modified by the loop body. So the third change of the generator is the usage of the difference between the upper bound and iteration number as the value of loop counter in the body of *rep*. In the case of iteration continuation, the loop counter is initialized by the difference between the same value and 1. In the case of the loop exit, the loop counter is not modified. Note that such approach can simplify the proof because it expresses the value of the loop counter explicitly.

4 Verification of Insertion Sort Program

Let us demonstrate the motivating example. Consider the following insertion sort program, which orders a given linear array a of the length n:

```
/* P */ void insertion_sort(int a[], int n) {int k, i, j;
        /* INV */ for (i = 1; i < n; i++) {
                        k = a[i];
                        for (j = i - 1; j >= 0; j--) {
                             if (a[j] <= k) break;
                             a[j + 1] = a[j];}
                        a[j + 1] = k;}}/* Q */
```

The program pre-condition, post-condition, and invariant have the form:

$$P \equiv 0 < n \wedge a = a_0 \wedge n \leq len(a_0),\ Q \equiv perm(0, n-1, a_0, a) \wedge ord(0, n-1, a),$$
$$INV \equiv i \leq n \wedge n \leq len(a) \wedge len(a_0) = len(a) \wedge$$
$$a_0[i : n-1] = a[i : n-1] \wedge perm(0, i-1, a_0, a) \wedge ord(0, i-1, a),$$

where $perm(i, j, a_0, a)$ means that array a is the permutation of array a_0 from i-th to j-th element, $ord(i, j, a)$ denotes that array a is ordered from i-th to j-th element. Note that Galeotti et al. [5] did not prove the permutation, and Srivastava et al. [15] used weaker property $\forall i \exists j\ (0 \leq i < n) \Rightarrow (0 \leq j < n \wedge a_0[i] = a[j])$.

Applying rules of the mixed axiomatic semantics [1] we obtain three verification conditions: the condition of loop entry, the condition of loop exit and the condition of iteration continuation.

The first and the second verification conditions were proved in ACL2 automatically. Consider the third verification condition, which is the most difficult. It has the following form:

$$i < n \wedge INV \Rightarrow ((((INV(i \leftarrow i+1))(a \leftarrow update\text{-}nth(j+1, k, a)))$$
$$(j \leftarrow rep(i, envir\text{-}init(i, k), frame\text{-}init(i-1, a)).j,$$
$$a \leftarrow rep(i, envir\text{-}init(i, k), frame\text{-}init(i-1, a)).a))(k \leftarrow a[i])),$$

Due to the symbolic method of definite iteration verification, we do not need to provide an invariant for inner loop. This verification condition can be found

in [17] (in the syntax of ACL2). It was automatically generated and named `vc-3` in the file `vc-3.lisp`. To prove this verification condition the strategies from section 5 were applied.

5 Method of Automation of Verification Conditions Proving

During this research, four strategies of verification conditions proving were developed. They are based on the automatic generation of lemmas. Their proof can help to prove the verification condition. Automatically generated formulas can be not theorems, therefore only successful proving of them in ACL2 allows adding them into the underlying theory. In ACL2, such formulas can be given to the user for proving them in interactive mode or can be proved automatically. We will give here key lemmas, which were added to the underlying theory in our example. These lemmas allowed ACL2 to prove the verification condition.

Let us introduce common notions for all strategies. Each of them gets a finite downward iteration over array a. We will define strategies using notions from section 3.

Let us consider the verification condition of a form $(X_1 \wedge X_2 \wedge \ldots \wedge X_w) \Rightarrow (C_1 \wedge C_2 \wedge \ldots \wedge C_m)$, where X_1, X_2, …, X_w are hypotheses and C_1, C_2, …, C_m are goals. If the verification is not of that form, let us bring it to such form. We will consider each goal separately: $(X_1 \wedge X_2 \wedge \ldots \wedge X_w) \Rightarrow C_i$, where $1 \leq i \leq m$. Let $Y = \{C_1, C_2, \ldots, C_m\}$.

Let us make a correspondence between parameters of strategies and our finite iteration with its continuation condition from our example. In our case, the parameter a stands for array a, parameter n stands for variable i, parameter i stands for variable j. Parameter $T \equiv INV \wedge (i < n)$. Let $a[i : j]$ be a subarray of array a from i-th to j-th element inclusively.

5.1 The Strategy of Premises Choice

The condition of the applicability of this strategy is the form of considered finite iteration (with possible loop exit). This strategy is applied, if during verification condition proving, we try to prove a statement about the property of *rep*. Let R be such a statement. Thereby, the first argument of our strategy is the finite iteration, the second one is the definition of *rep*, the third argument is R.

The strategy is oriented to solving a problem of transformation of R to lemma, which has the form of implication. R becomes the conclusion of such implication. Therefore, the problem is reduced to a generation of the premise, which should allow to prove the lemma and the verification condition.

To overcome this difficulty we use more generalized statements as premises than T. For example, the statement $L_1 \equiv (\mathit{iteration} \in N) \wedge (\mathit{iteration} \leq \mathit{env.upper\text{-}bound}) \wedge (\mathit{env.upper\text{-}bound} < (\mathit{len}(fr.a))) \wedge (fr.j = (\mathit{env.upper\text{-}bound} - 1))$ or the statement $L_2 \equiv (\mathit{env.upper\text{-}bound} \in N) \wedge (\mathit{env.upper\text{-}bound} < (\mathit{len}(fr.a))) \wedge (fr.j = (\mathit{env.upper\text{-}bound} - 1)) \wedge \neg fr.\mathit{loop\text{-}break}$.

We plan to extend the set of premises. The choice is determined by one, which helps ACL2 to prove the lemma. For each $L \in \{L_1, L_2\}$, we try to prove formula $lm \equiv L \Rightarrow P$ in ACL2. All such lemmas start with the prefix `rep-lemma-` in [17].

Let us generate auxiliary formula lm' in a form of implication. Its premise is T and its conclusion is constructed from R by replacing n, env, and fr by iteration parameters. Let us substitute the iteration parameters into lm to prove lm'. Because of the same premises, it is more convenient to use lm' for verification condition proving. In [17] the names of all such lemmas start with prefix `vc-3-lemma-`.

For example, consider the statement about equality of two subarrays as $R \equiv a[0 : rep(iteration, env, fr).j] = (rep(iteration, env, fr).a)[0 : rep(iteration, env, fr).j]$. Applying our strategy to R, we add a lemma $L_1 \Rightarrow P$ to the underlying theory, which can be found in [17] under the name `rep-lemma-76`.

5.2 The Strategy for Finite Iteration Over Changeable Array

The condition of the applicability of this strategy is the form of considered finite iteration and the presence of assignment statement `a[expr-index] = expr-value;` in a loop body. Let an iteration consists of w assignment statements. With the help of the function *c_kernel_translator*, we will translate each expression $expr\text{-}index_i$ to the expression $expr\text{-}ind_i$ of ACL2 language for each i: $1 \leq i \leq w$.

Let us generate and try to prove by the strategy from 5.1 the following statement: $(index \neq expr\text{-}ind_1) \wedge \ldots \wedge (index \neq expr\text{-}ind_w) \Rightarrow rep(iteration - 1, env, fr).a[index] = rep(iteration, env, fr).a[index]$. In case of a successful proof of such formula, the corresponding lemma is added to the underlying theory. It states, that an array element, whose index is not in the set of indices of left operands of assignment statements, is not changed at the next iteration.

Consider the application of this strategy to our example. Note that the loop contains the assignment statement `a[j + 1] = a[j];`. Since the value of loop counter is $env.upper\text{-}bound - iteration$, the generated lemma `rep-lemma-22` [17] has the following form: $L_1 \wedge (index \neq (env.upper\text{-}bound - iteration) + 1) \Rightarrow rep(iteration - 1, env, fr).a[index] = rep(iteration, env, fr).\ a[index]$.

5.3 The Strategy for Finite Iteration With `break` Statement

The condition of the applicability of this strategy is the form of considered finite iteration and the presence of `break` statement. Let break-condition be a conjunction of controlling expressions of the `if` statements on the path to `break` statement (we make all necessary substitutions in case of assignment statements). In fact, break-condition is a function $br\text{-}cond(iteration, env, fr)$.

The value of the field *loop-break* of the structure $frame$ is the detector for certain iteration, whether `break` statement occurred earlier. Note that the number of iteration, which led to loop exit, is $br\text{-}iter = env.upper\text{-}bound -$

$rep(env.upper\text{-}bound, env, fr).i$. This strategy attempts to prove a set of auxiliary lemmas about the execution of `break` statement:

1. If $rep(iteration, env, fr).loop\text{-}break$ then $rep(iteration, env, fr).i = rep($ $iteration - 1, env, fr).i$.

2. If $\neg rep(iteration, env, fr).loop\text{-}break$ then $rep(iteration, env, fr).i = env.$ $upper\text{-}bound - iteration - 1$.

3. If $rep(iter, env, fr).loop\text{-}break$ and $iter \leq iteration$ then $rep(iteration,$ $env, fr).loop\text{-}break$.

4. If $rep(iter, env, fr).loop\text{-}break$ and $iter \leq iteration$ then $rep(iter, env, fr)$ $= rep(iteration, env, fr)$.

5. If $\neg rep(iteration, env, fr).loop\text{-}break$ and $iter \leq iteration$ then $\neg rep(iter,$ $env, fr).loop\text{-}break$.

6. $\neg(br\text{-}iter - 1, fr, env).loop\text{-}break$ and $rep(br\text{-}iter - 1, fr, env).loop\text{-}break$.

7. If $iter \in [br\text{-}iter : env.upper\text{-}bound]$ then $rep(env, iter, fr).loop\text{-}break$.

8. If $iteration \in [0 : br\text{-}iter - 1]$ then $\neg rep(iteration, env, fr).loop\text{-}break$.

9. If $iteration \in [0 : br\text{-}iter - 1]$ then $\neg br\text{-}cond(iteration, env, fr)$.

10. $iteration \in [br\text{-}iter : env.upper\text{-}bound] \Rightarrow br\text{-}cond(iteration, env, fr)$.

11. $\neg br\text{-}cond(br\text{-}iter - 1, env, fr)$ and $br\text{-}cond(br\text{-}iter, env, fr)$.

These statements are based on the fact, that the property "whether a loop exit occurred" is monotonic against the number of iteration. These statements are handled by a strategy described in 5.1.

Consider the application of this strategy to our example. As the loop contains `break` statement the break-condition is $(a[rep(iteration-1, env, fr).j]) \leq env.k$. In our case the statement 11 has the form: $a[rep(br\text{-}iter - 1, env, fr).j] > env.k$ and $a[rep(br\text{-}iter, env, fr).j, env, fr)] \leq env.k$. Using obtained by the strategy statements 1 and 2 we have the equivalent break-condition: $a[rep(env.upper\text{-}bound, env, fr).j + 2] > env.k$ and $a[rep(env.upper\text{-}bound, env, fr).j] \leq env.k$. By the strategy from 5.1 these statements are transformed to lemmas [17] `rep-lemma-83`: $L_2 \Rightarrow env.k < a[rep(env.upper\text{-}bound, env, fr).j+2]$ and `rep-lemma-108`: $L_2 \Rightarrow a[rep(env.upper\text{-}bound, env, fr).j] \leq env.k$.

5.4 The Strategy for Functions With Concatenation Property

The condition of the applicability of this strategy is the form of considered finite iteration (with possible loop exit) and the presence of predicates satisfying concatenation or concatenation with a splice at bounds property.

The predicate V has the concatenation property if $(V(i, k, u, \ldots) \wedge V(k, j, u,$ $\ldots) \wedge (i \leq k) \wedge (k < j)) \Rightarrow V(i, j, u, \ldots)$. The predicate V has the concatenation with the splice at bounds property if $(V(i, k, u, \ldots) \wedge V(k, j, u, \ldots) \wedge (i \leq k) \wedge (k <$ $j) \wedge f(u[k], u[k + 1])) \Rightarrow V(i, j, u, \ldots)$.

These property patterns are used in check whether this strategy is applicable. To do this, loop and post-condition analysis are performed. For all predicates in post-condition, we search theorems satisfying given property patterns in all used theories.

Loop analysis allows ascertaining whether the loop corresponds to the form of considering finite iteration. Thus, the first argument of our strategy is finite

iteration, the second argument is the definition of rep, the third one is the verification condition. The strategy starts at an analysis of the elements of the set Y.

Let Z be a set of goals from Y, which are the applications of a predicate satisfying concatenation or concatenation with a splice at bounds property: $Z \subset Y$. For each goal $\phi \in Z$ the following steps are performed:

1. Let $\phi \equiv U(\ldots)$. Consider splits of arrays a and $rep(n, env, fr).a$. Let us apply the strategy from 5.1 to $a[0 : rep(n, env, fr).i] = (rep(n, env, fr).a)[0 : rep(n, env, fr).i]$. The segment $[0 : rep(n, env, fr).i]$ is not covered by iteration, therefore, we can suppose that it is not modified by rep. If the lemma was added to the underlying theory we try to prove $T \wedge U(0, rep(n, env, fr).i) \Rightarrow U(0, rep(n, env,\ fr).i, rep(n, env, fr).a, \ldots)$. In case of successful proof, we add this formula to the underlying theory.

2. In case of presence of statements `a[i + expr] = a[i];`, where $expr$ is a C-light expression, the hypothesis about array shift arises. With the help of $c\_kernel\_translator$ from [8] we obtain $expression$, which is $expr$ in ACL2 language. We apply a strategy from 5.1 to $a[(rep(n, env, fr).i + 1) : (n - expression)] = (rep(n, env, fr).a)[(rep(n, env,\ fr).i + 1 + expression) : n]$. Thereby, after loop exit we check the coincidence of initial subarray and shifted by rep one. If the lemma was added to the underlying theory, we try to prove $T \wedge U(rep(n, env, fr).i + 1, n - expression, a, \ldots) \Rightarrow U(rep(n, env, fr).i + 1 + expression, n, rep(n, env, fr).\ a, \ldots)$. In case of successful proof, we add this formula to the underlying theory.

3. If predicate U satisfies the property of concatenation with a splice at bounds by condition f, then we apply the strategy from 5.3. Note that break-condition can contain f and depends on $br\text{-}iter$ defined in 5.3. If we proved the formula obtained from the statement 11 of the subsection 5.3, then such lemma can help to prove the range splice. We try to prove $T \wedge \neg br\text{-}cond(br\text{-}iter, env, fr) \wedge br\text{-}cond(br\text{-}iter, env, fr) \Rightarrow f((rep(n, env, fr).a)[rep(n, env, fr).i], (rep(n, env, fr).a)[rep(n, env, fr).i + 1])$.

This lemma states about the range splice between $rep(n, env, fr).i$ and $rep(n, env, fr).i + 1$, i. e. in the point of loop exit. In case of successful proof, we add this formula to the underlying theory.

For statements of the form `a[i + expr] = a[i];` we generate and check the formula about range splice of $rep(n, env, fr).i + expression$ and $rep(n, env, fr).i + expression + 1)$.

4. Apply added to the underlying theory lemmas to prove $T \Rightarrow \phi$.

Let us apply this strategy to our example. After post-condition analysis, the theorems `permutation-7` and `ordered-3` were detected in the underlying theory. Since `permutation-7` satisfies the pattern, the predicate $perm$ satisfies concatenation property. The theorem `ordered-3` satisfies the pattern with a splice at bounds, where $\leq$ is used as a relation f. Therefore, the predicate ord satisfies the concatenation property with a splice at bounds with respect to $\leq$.

Consider lemmas appeared at proving two goals. Let A be the goal containing predicate $perm$. Let B be the goal containing predicate ord. Thus,

$A, B \in Z$. Then, let $G \equiv T \Rightarrow A$ and let $H \equiv T \Rightarrow B$. Let $e = update\text{-}nth(rep(i, env, fr).j + 1, k, rep(i, env, fr).a)$.

Consider first the application of strategy steps to formula A. The application of strategy to $a[0 : rep(n, env, fr).j] = (rep(i, env, fr).a)[0 : rep(i, env, fr).j]$ is described in 5.1. After that, the permutation of a and e in the range $[0 : rep(i, env, fr).j]$ was proved.

During analysis, the statement `a[j + 1] = a[j];` was detected, which is a potential array shift. Thus, the constant 1 corresponds to *expression*. With the help of `rep-lemma-22` and the strategy from 5.1, `rep-lemma-55` was obtained from $a[rep(i, env, fr).j + 1 : i - 1] = (rep(i, env, fr).a[rep(i, env, fr).j + 2 : i]$. It allowed to prove permutation of $a[rep(i, env, fr).j + 1 : i - 1]$ and $e[rep(i, env, fr).j + 2 : i]$.

The statement $e[rep(i, env, fr).j + 1 : rep(i, env, fr).j + 1] = a[i : i]$ was proved automatically. It allowed to prove the permutation of $a[i : i]$ and $e[rep(i, env, fr).j + 1 : rep(i, env, fr).j + 1]$. As permutation satisfies concatenation property, the permutation of a and e in the range $[rep(i, env, fr).j + 1 : i]$ was proved. Finally, using concatenation property we get permutation of a and e in the range $[0 : i]$.

Consider the application of strategy steps to formula B. The predicate *ord* satisfies the property of concatenation with a splice at bounds, so it is necessary to check that the relation $\leq$ holds at bounds. This property was successfully checked in 5.3.

All mentioned lemmas and the full proof can be found in [17].

6 Conclusion

This paper represents the method for the automation of the C-light program verification. In case of definite iteration over changeable arrays with loop exit, this method allows generating verification conditions without loop invariants.

This generation is based on the new inference rule for the C-light `for` statement which uses the replacement operation. It is generated automatically by the special algorithm [8], which translates loop body statements to ACL2 constructs. In this paper, we described changes to this algorithm, which extends the application of our method to downward iterations.

To prove obtained verification conditions, we apply special strategies based on lemmas generation. The successful proving of such lemmas can help to prove the verification conditions. We developed four strategies. Their application was illustrated by the verification of insertion sort program. They supplement the symbolic method of finite iterations and allow automatizing the process of deductive verification.

Also, the verification of the functions implementing BLAS interface [3] is an important problem. Earlier we performed such experiments successfully [7]. Our methods allowed us to verify the function *asum*, which implements the corresponding function from BLAS interface: it calculates the sum of absolute values of a vector.

References

1. Anureev, I. S., Maryasov, I. V., Nepomniaschy, V. A.: C-programs Verification Based on Mixed Axiomatic Semantics. Automatic Control and Computer Sciences **45**(7), 485–500 (2011)
2. Cohen, E., Dahlweid, M., Hillebrand, M., Leinenbach, D., Moskał, M., Santen, T., Schulte, W., Tobies, S.: VCC: A Practical System for Verifying Concurrent C. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009, LNCS, vol. 5674, pp. 23–42. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03359-9_2
3. Dongarra, J. J., van der Steen, A. J.: High-performance Computing Systems: Status and Outlook. Acta Numerica **21**, 379–474 (2012)
4. Filliâtre, J.-C., Marché, C.: Multi-prover Verification of C Programs. In: Davies, J., Schulte, W., Barnett, M. (eds.) ICFEM 2004, LNCS, vol 3308, pp. 15–29. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30482-1_10
5. Galeotti, J. P., Furia, C. A., May, E., Fraser, G., Zeller, A.: Inferring Loop Invariants by Mutation, Dynamic Analysis, and Static Checking. IEEE Transactions on Software Engineering **41**(10), 1019–1037 (2015)
6. Kaufmann, M., Moore, J. S.: An Industrial Strength Theorem Prover for a Logic Based on Common Lisp. IEEE Transactions on Software Engineering **23**(4), 203–213 (1997)
7. Kondratyev, D.: Implementing the Symbolic Method of Verification in the C-Light Project. In: Petrenko, A., Voronkov, A. (eds.) PSI 2017, LNCS, vol. 10742, pp. 227–240. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74313-4_17
8. Kondratyev, D. A., Maryasov, I. V., Nepomniaschy, V. A.: The Automation of C Program Verification by Symbolic Method of Loop Invariants Elimination. Modeling and Analysis of Information Systems, **25**(5), 491–505 (2018) (In Russian)
9. Kovács, L.: Symbolic Computation and Automated Reasoning for Program Analysis. In: Abraham, E., Huisman, M. (eds.) IFM 2016, LNCS, vol. 9681, pp. 20–27. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33693-0_2
10. Li, J., Sun, J., Li, L., Le, Q. L., Lin, S.-W.: Automatic Loop Invariant Generation and Refinement through Selective Sampling. In: Proceedings on ASE 2017, pp. 782–792. Conference Publishing Consulting, Passau (2017).
11. Maryasov, I. V., Nepomniaschy, V. A.: Loop Invariants Elimination for Definite Iterations Over Unchangeable Data Structures in C Programs. Modeling and Analysis of Information Systems, **22**(6), 773–782 (2015)
12. Maryasov, I. V., Nepomniaschy, V. A., Kondratyev, D. A.: Invariant Elimination of Definite Iterations Over Arrays in C Programs Verification. Modeling and Analysis of Information Systems, **24**(6), 743–754 (2017)
13. Maryasov, I. V., Nepomniaschy, V. A., Promsky, A. V., Kondratyev, D. A.: Automatic C Program Verification Based on Mixed Axiomatic Semantics. Automatic Control and Computer Sciences, **48**(7), 407–414 (2014)
14. Nepomniaschy, V. A.: Symbolic Method of Verification of Definite Iterations Over Altered Data Structures. Programming and Computer Software, **31**(1), 1–9 (2005)
15. Srivastava, S., Gulwani, S., Foster, J. S.: Template-based Program Verification and Program Synthesis. International Journal on Software Tools for Technology Transfer, **15**(5-6), 497–518 (2012)
16. Tuerk, T.: Local Reasoning About While-loops. In: Theory Workshop Proceedings on VSTTE 2010, pp. 29–39. Heriot-Watt University, Edinburgh (2010)
17. Verification of Insertion Sorting Program, https://bitbucket.org/Kondratyev/sorting. Last accessed 26 Apr 2019

Automated Sisal program verification with ACL2*

Dmitry Kondratyev and Alexei Promsky

A.P. Ershov Institute of Informatics Systems, 630090, Novosibirsk, Russia
apple-66@mail.ru, promsky@iis.nsk.su

Abstract. The Sisal programming environment which is being developed in IIS also includes a verification module. The previously developed C-light verification system serves as its base, since the C language representations of Sisal programs are actually processed. The preservation of equivalence during translation provides correctness of this two-stage verification. At the moment we concentrate our efforts on verification of Sisal loop expressions which are translated into the C `for`-loops. Trying to avoid the well-known problem of the loop invariants we apply a symbolic method of definite iterations. This technique expresses the loop effect in symbolic form. However, the Sisal loop expressions sometimes lead to peculiar C loops. The symbolic forms of such loops in verification conditions are too complex to be proved automatically. In this paper we represent a proof strategy for such formulas. Our strategy introduces logical formula transformations which, in general, do not maintain equivalence. However, the truth of resulting formula guarantees truth of the original one. We also describe here a verification example.

Keywords: automated theorem proof · C-light · Sisal · deductive verification.

1 Introduction

Programming environment for the Sisal language [11, 14] is one of the urgent projects in IIS. It aims mainly at efficiency, so the input program is translated into an intermediate form which, in turn, can be aggressively optimized [13]. In addition, intermediate form can be translated into the C language[9, 12].

A more recent feature of the project relates to deductive program verification [1], which traditionally rests on axiomatic semantics and generation of verification conditions. However, at the moment we do not have a Hoare's logics for Sisal. And this is where translation into the C comes in handy. Another actual project of IIS is the C-light verification system. Despite its name, the C-light is a quite representative subset of the Standard C with a full operational semantics. This project also involves a two-stage scheme, thus introducing a special core, the C-kernel language, which possesses a sound axiomatic semantics. Details can be

* This research is being supported by grant No. 18-11-00118 from Russian Science Foundation.

found in [23, 25]. Trying to mechanize the verification proofs we experiment with popular tools. In this paper we address the interactive theorem prover ACL2[15].

Among the traditional woes of deductive verification, loop invariants begin to play a crucial role here. Originally, loop invariants together with pre- and postconditions are provided by user prior to verification process [1]. But the loop expressions of Sisal are translated on the fly into the C `for`-loops, thus requiring automatic generation of appropriate invariants. This is a very complex task, though some successes were demonstrated by researchers [28, 20]. To avoid it we actually apply to the third actively developing project of our institute. The symbolic method of verification of definite iterations over altered data structures introduces a special replacement operation `rep`[24, 16]. As the name suggests the idea is to represent the loop action in some symbolic form.

The recursive nature of `rep` requires induction over iteration numbers. Our initial attempts to validate such induction in ACL2 were unsuccessful. So we need to develop automatic proof strategies. Some of them were invented in adjacent experiments. For example, a strategy from [22] defines lemmas satisfying certain restrictions which, in turn, depend on verification condition structure. Unfortunately, that strategy is only interactive. In this paper we represent a more recent strategy which allows a fully automated proof in ACL2.

Related work. The attempts to develop an efficient Sisal compiler are well-known. For example, the paper [4] describes Optimizing Sisal Compiler (OSC) which also produces the C code.

The intermediate languages in program verification have become a standard de facto nowadays [29, 18]. Among the most popular languages we can note Boogie [19] and WhyML [8]. The first one is used in such systems as AutoProof [27], HAVOC[2], VCC [5], Spec# [3]. The latter (together with language Jessie [21]) serves as a base in Caduceus [6], Jessie [21], Krakatoa [7].

The system `ACL2(ml)` [10] bases its proof strategies on combination of two methods. The first one uses statistical machine learning to recognize proof patterns. The second one is a symbolic finding of analogous lemma. However, the specific domain theories may vary drastically depending on programs. Thus, machine learning methods are not ideal for verification condition proofs.

The rest of this paper is organized as follows. Section 2 provides some preliminary information over symbolic verification method (Subsection 2.1), the Sisal language and ACL2 (2.2), as well as a program example (2.3). A new proof strategy for loops emerging due to Sisal loop expressions is represented in Section 3. An application of this strategy to our example forms the Section 4. Section 5 concludes.

2 Preliminary information

The paper volume does not provide enough space for detailed notations. Reader can find an extended version of the paper in our on-line repository [17].

2.1 Symbolic method of verification of definite iterations

The general representation of definite iteration over data structure takes the following form:

$$\textbf{for x in S do v} := \textbf{body}(\textbf{v}, \textbf{x}) \textbf{ end}$$

Here S is a data structure, x is variable of type *element of* S, v is a tuple of loop variables (excluding x) and *body* represents calculations within the loop which do not alter x. The loop body only allows assignments, `if` statements (including nested ones) and `break` statements. We denote the tuple of all elements of S as $vec(S)$. Suppose that $vec(S) = [s_1, s_2, \ldots, s_n]$. Then the loop body consecutively iterates in such manner that x equates to $s_1, s_2, \ldots, s_n$ and $body(v, s_j)$ can modify $s_1, s_2, \ldots, s_{j-1}$.

The main advantage of this approach reveals itself when it comes to the proof of Hoare triples. Not only it does not require some loop invariant, but also it does not split the proof tree. It simply introduces a linear replacement operation $rep(v, s, body)$ which expresses the value of v after iteration:

$$\frac{\{P\}\,\mathbf{A};\{Q(v \leftarrow rep(v, S, body))\}}{\{P\}\,\mathbf{A};\ \textbf{for x in S do v} := \textbf{body}(\textbf{v}, \textbf{x})\,\textbf{end}\{Q\}}$$

Here **A** is a context (i.e. all statements preceding iteration), $\leftarrow$ denotes substitution for all free occurrences, P and Q are pre- and postcondition respectively.

Since the C is our intermediate language, we can consider a special case if definite iterations in the corresponding syntax. Given S is an array of length n and $S \in v$, it looks like

$$\textbf{for}\,(\mathbf{i} = \mathbf{0}; \mathbf{i} < \mathbf{n}; \mathbf{i}++)\ \mathbf{v} := \textbf{body}(\mathbf{v}, \mathbf{i})\ \textbf{end},$$

where **v := body(v, i)** consists of assignments, `if` statements and `break` statements. Only expressions without side-effects are allowed at right hand sides of assignments and in conditions of `if`s.

Thus, v is a tuple of all objects that can be altered in loop body. Suppose v_0 is a value of v before the loop and v_i denotes the value of v after ith iteration. Then $v_i = body(v_{i-1}, s_i)$ for $i = 1, ..., n$.

Now we define the replacement operation. Let $rep(0,\ v_0) = v_0$ and $rep(i,\ v_0) = body(rep(i-1,\ v_0), s_i)$. Thus, the main property of this operation is $rep(i,\ v_0) = v_i$. The recursive definition of *rep* is built automatically by analysis of *body* [22].

If `break` was executed during iteration j $(0 < j \leq n)$ we model it as if iterations still go on, but the value of v does not change. I.e. for all k such that $j \leq k \leq n$ $rep(j,\ v_0) = rep(k,\ v_0)$.

So the corresponding proof rule takes the form

$$\frac{\{P\}\,\mathbf{A};\{Q(v \leftarrow rep(n,\ v_0))\}}{\{P\}\,\mathbf{A};\ \textbf{for}\,(\mathbf{i} = \mathbf{0}; \mathbf{i} < \mathbf{n}; \mathbf{i}++)\ \mathbf{v} := \textbf{body}(\mathbf{v}, \mathbf{i})\ \textbf{end}\{Q\}}$$

Since the replacement operation returns vector v of length m, it may be appropriate to define m atomic replacement operations.

2.2 The Sisal language and ACL2

Let us limit ourselves to survey of those parts of languages which are necessary to understand the remaining sections. Both Sisal and ACL2 follow the functional programming style allowing to avoid side-effects. But their purpose differs, Sisal aims at efficient calculations, whereas ACL2 is suitable to define logical theories.

The data structure handling in Sisal is based on loop expressions [11]. Their form serves the automatic vectorization in some cases. The loop expression combines three parts: loop control, loop body and returns clause. The heading loop control (denoted by keyword `for`) declares variables and their ranges during iterations. Loop body consists of expressions modifying loop variables. The operator `old` takes a variable name as an argument and returns its value from the previous iteration. The control expressions (the keyword `while`) can also take place. The falsity of such expression results in abrupt termination. The returns clause is a list of reductions. Every loop variable is associated with a reducible sequence of values corresponding to iterations. Reductions allow us to apply certain operations to those sequences. For example, reduction `value` produced the last value of a reducible sequence. This list of reduction itself is a value of whole loop.

On the contrary ACL2 prefers recursion to manipulate data structures. Since technically ACL2 is a CLisp library it uses the standard `defun` for function definitions. Within function definition a user can introduce construction `declare` which, in turn, can contain sections `xargs` and `measure`. As long as we try to conduct some inductive proofs in ACL2 these sections can be useful, especially when termination of recursive function is under concern.

The type checks in our examples will use special predicates. For instance, `integerp` tests whether its argument is an integer, `zp` match its argument against zero and `integer-listp` is satisfiable only by lists of integers.

We use lists to model Sisal arrays as well as intermediate C arrays. The function `nth` accesses list elements while `length` returns the number of list elements.

The logical connectives in ACL2 are denoted by obvious `not`, `and`, `or` and `implies`. The statement `defthm` serves as theorem definition. It can contain special sections. For example, section `hints` introduces "directions" which ACL2 should follow during proof. Such recommendations can be defined for every formula that contributes to the theorem proof. These formulas are called "goals" and the theorem itself is a goal (denoted by keyword `"Goal"`). Finally, the hint `induct` advises ACL2 to apply an induction scheme when it tries to satisfy a goal.

2.3 Study case

Consider the following Sisal program counting occurrences of some key in array:

```
function search_count (a: array of integer,
                       n, entr key: integer returns integer)
    for cnt := 0, result := 0; i in 1..n
```

```
    while !(cnt = entr) do
        cnt := if a[i]=key then old cnt + 1;
        result := if cnt=entr then 1;
    returns value of result
    end for
end function
```

Its precondition (ACL2 syntax) looks like

```
(and (integer-listp a)
     (integerp n)
     (integerp key)
     (< 0 n)
     (<= n (length a))
)
```

whereas its postcondition is

```
(and (implies (=> entr (cnt 0 (- n 1) key a))
              (equal result 1)
     )
     (implies (< entr (cnt 0 (- n 1) key a))
              (equal result 0)
     )
)
```

The logical function `cnt` computes the number of elements equal to `key` in the sublist from the i-th up to the j-th element. The complete definition of `cnt` can be found in repository [17].

The translation stage leads to the following C program:

```
int search_count(int* arr, int length, int key, int entr)
{
    auto int result = 0, cnt = 0, i;
    for (i = 0; i < length; i++)
    {
        if (key == arr[i]) cnt++;
        if (cnt == entr) {result = 1; break;}
    }
    return result;
}
```

A traditional verification condition (VC) generator would produce five VCs due to one loop and two `if` statements. On the contrary, our VCG uses the symbolic method deriving single (though complex) VC. Its complete definition in ACL2 can also be found in [17]. In the rest of the paper we apply to a simplified scheme of VC.

As we mentioned above, the symbolic method introduces logical functions representing substitution operation for the mutable objects within loop body

[22]. For the program under discussion function `rep1` symbolically reflects changes of variable `cnt` on each iteration:

```
(defun rep1(i key entr a)
    (if
        (zp i)
        0
        (if
            (= entr (rep1 (- i 1) key entr a))
            (rep1 (- i 1) key entr a)
            (if
                (= key (nth (- i 1) a))
                (+ 1 (rep1 (- i 1) key entr a))
                (rep1 (- i 1) key entr a)
            )
        )
    )
)
```

while `rep2` embodies the effect of assignments to `result`:

```
(defun rep2(i key entr a)
    (if
        (zp i)
        0
        (if
            (= entr (rep1 i key entr a))
            1
            (rep2 (- i 1) key entr a)
        )
    )
)
```

The replacement operation is always defined by recursion on the first argument which corresponds to the iteration number. So, VCs containing replacement operation require induction. In our experiments we use induction on the data structure length.

However, ACL2 fails to prove such VC when it uses solely induction on `n`. This answer of ACL2 was analyzed automatically resulting in automatic application of the proof strategy from the following Section.

3 The proof strategy

Now, let us see what hints can be given to ACL2 in order to achieve the goal. The method we describe here can be used in ACL2 to automatically prove VCs inherent in programs that deal with definite iterations.

The idea is as follows: for a given VC ϕ we construct a logically stronger formula θ (though inequivalent in general case). Since $\theta \to \phi$ by construction, it is enough to validate θ. Some considerations must be taken into account before we start. First, by construction a typical VC ϕ is a propositional combination of equality/inequality terms. Apart from functions of a domain theory, VCs contain special functions rep_i mentioned above. Each rep_i corresponds to its own program variable that can be altered during loop execution. The length of a processed sub-array plays an important role here because it serves as induction variable. The initial values of program variables and loop exit conditions also can be used in definitions if rep_i.

The VC proof algorithm itself is based on a stepwise transformation of formula ϕ. Every local rewriting gives a stronger (perhaps nonequivalent) formula. When we have a choice, we prefer rewritings allowing to avoid problems in ACL2. For example, an in-line substitution of a non-recursive function body takes place instead of its invocation.

The arguments of algorithm are formula ϕ, the sub-array length n, functions rep_i (let k be their quantity), an underlying theory, initial values of program variables (at loop entry point) and loop exit condition. Underlying theory includes definitions of functions whose calls are sub-formulas in ϕ as well as theorems about these functions.

The result of this algorithm can be "formula ϕ is valid" if ACL2 has succeeded in proof of a stronger formula θ or "unknown" otherwise.

Our algorithm consists of the following six steps:

1. Formula ϕ is being converted into equivalent clause conjunction.
2. For every clause we construct a graph of relations between variables and functional calls in the clause premise. So, variables and function calls are the nodes. As long as clause premise is a conjunction of hypotheses we analyze them to establish edges. Namely, the nodes a and b are joined by the edge (a, b) with a label R where $R \in \{=, \neq, <, >, \leq, \geq\}$ iff either a or b is variable and hypothesis $R(a, b)$ exists in clause premise.
   For every clause, for every variable v and for every relation $R \in \{=, \neq, <, >, \leq, \geq\}$ we define a special procedure which searches for the (nearest or corresponding) function call and a list of hypotheses validating that call. The searching procedure takes the relation graph G of considered clause, variable v and relation R as its arguments. The returned value can either be message "corresponding function call not found" or a function call accompanied by hypotheses list. Depending on relation R this procedure can be defined as follows:
   (a) if R is "=", then let F be a set of nodes in G which are reachable from v by transitive closure $=^*$;
   (b) if R is $\leq$, then let F be a set of nodes reachable from v by $(= \cup < \cup \leq)^*$ (i.e. we traverse all edges labeled by either $=$ or $<$ or $\leq$);
   (c) if R is $\geq$ then F combines all nodes reachable from v by $(= \cup > \cup \geq)^*$;
   (d) if R is $<$ then F is a set of all nodes reachable from v by relation

$$(= \cup < \cup \leq)^* \circ (<) \circ (= \cup < \cup \leq)^*$$

(we traverse all edges labeled by $=$, $<$ or $\leq$ and one of them must be labeled by $<$);

(e) if R is $>$ then let F be a set of nodes reachable from v by $(= \cup > \cup \geq)^* \circ (>) \circ (= \cup > \cup \geq)^*$ (by analogy with (e) one of the edges in the path must be $>$);

(f) if R is $\neq$ we define F as all nodes reachable from v by $(=)^* \circ (\neq \cup < \cup >) \circ (=)^*$ (again, there must be an edge labeled by $\neq$ or $<$ or $>$);

Finally, if F contains at least one function call then procedure returns the nearest one (by amount of used hypotheses) as well as a list of equalities used along the corresponding path. Otherwise, procedure signals `"corresponding function call not found"`.

Now that relation graph has been constructed for a clause, we begin to process the conclusion of clause. Conclusion is a disjunction of goals, each of them looks like $R(c, d)$ where $R \in \{=, \neq, <, >, \leq, \geq\}$, c and d are either constants, variables or function calls. For a given $R(c, d)$ we introduce auxiliary variables v and w assigning them values of c and d respectively. Another pair of auxiliary variables q and r is initialized with empty lists. In case that the first argument c of relation R is a variable, the searching procedure starts. If search is successful the discovered function call and list conjuncts are assigned to variables v and q respectively. Next we analyze the second argument d of relation R. If v is equal to c or R is not "$\neq$" and d is a variable then the search procedure looks for corresponding function call:

• if relation R is either "$=$" or "$\neq$" then d and R are passed to the procedure as its arguments;

• if relation R is either $<$ or $\leq$ then d and an "opposite" relation ($>$ or $\geq$ respectively) are passed to the procedure as its arguments;

• if relation R is either $>$ $\geq$ then d and an "opposite" relation ($<$ or $\leq$ respectively) are passed to the procedure as its arguments.

In case of success the corresponding function call and conjunct list are assigned to variables w and r respectively.

If the initial value of either v or w was changed the goal $R(c, d)$ is replaced by goal $(= \ v \ w)$ within conclusion of the clause under consideration. In the meantime, if R is $=$ the premise of the clause is being stripped of all hypotheses that occur in at least one of the lists q or r.

3. All rep_i that admit non-recursive redefinition are submitted to explicit substitution. It is sufficient to demonstrate that when the loop-exit condition is false rep_i is equal to initial value of the corresponding program variable. For every such function we create a tree representing its body. The internal nodes of such tree are `if` statements and leaves are values returned by function. The left descendant of a statement is its value when condition is true. The corresponding edge is labeled by condition of the statement. Correspondingly, the right descendant becomes the value when condition does not hold. The edge is labeled by negation of condition then.
4. For every clause we process its conclusion which in effect is a disjunction of individual goals. Let g be one of goals while c is an application of a non-recursive function occurring in g. We replace g by a conjunction of special

implications. First, let us consider the set of interim implications. Every interim implication corresponds to a leaf in function tree. Its premise represent conjunction of all edge labels on the path from root to that leaf. Its conclusion is the goal g in which every occurrence of c is replaced by leaf-value. For every interim implication the replacement procedure substitutes the actual arguments from invocation of c instead of variables within function body, thus transforming interim implication into the special one. After every substitution each conclusion needs to be transformed to fit the clause form. This step repeats as long as conclusions in clauses contain applications of non-recursive functions.

5. If the previous steps (1)–(4) resulted in modification of the formula we repeat them again.
6. Finally, ACL2 is applied to prove the formula by induction. Depending on its verdict the answer of the whole algorithm is either "formula ϕ is valid" or "unknown".

One of the advantages of this algorithm is possibility to fully automate it. We believe it can be generalized even further to be applied in other theorem provers. The adaptation for SMT-solvers CVC4 and Z3 is one of our future plans.

4 Application of the proof strategy

Let us consider how the strategy can be applied to verification of our study case (Sect.2.3). Note that strategy consistes of a sequence of steps (perhaps, reiterative).

Let A stand for the formula

```
(and (integerp n)
     (integerp key)
     (integerp entr)
     (integer-listp a)
)
```

Conjuncts in A state that variables n, key, $entr$ are integers and a is an integer list.

Let B denote formula

```
(and (< 0 n)
     (<= n (length a))
     (< 0 entr)
)
```

with an obvious meaning.

The formula C of the form

```
(<= entr (cnt 0 (- n 1) key a))
```

states that the number of occurrences of key within subsequence of a in between indices 0 and $n-1$ is not less than $entr$.

Another abbreviation D stands for

```
(= (rep2 n key entr a) 1)
```

meaning that the variable $result$ is equal to 1 just after execution of a loop which can be modeled by invocation of function $rep2$ with arguments n, key, $entr$ and a.

The formula E

```
(> entr (cnt 0 (- n 1) key a))
```

obviously is negation of C.

The formula F

```
(= (rep2 n key entr a) 0)
```

is a counterpart of D corresponding to unsuccessful search for key during loop execution.

The formula J

```
(= entr (rep1 n key entr a))
```

expresses equality of values of $entr$ and cnt just after execution of the loop. According to the symbolic verification method this execution is modeled by invocation of function $rep1$ with arguments n, key, $entr$ and a.

Finally, the formula K of the form

```
(zp n)
```

states that n is an integer greater than zero.

Now we aggregate these formulas into bigger ones. Let $L \equiv A \wedge B \wedge C$ and $M \equiv A \wedge B \wedge E$. These formulas do not reflect data structures that are being handled by algorithm, we only use them for simplicity.

Just before the step (1) of our algorithm the VC ϕ corresponds to the following pattern:

$$A \Rightarrow (B \Rightarrow ((C \Rightarrow D) \wedge (E \Rightarrow F))).$$

Let us omit the detailed description of results produced by each step (1)–(4). Reader can address online repository [17] to see the complete proof protocol. Enough to say that after single iteration of those steps we obtain the formula ϕ':

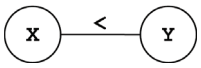
$$\begin{gathered}(L \Rightarrow (\neg K \vee (0 = 1))) \wedge \\ (L \Rightarrow ((K \vee \neg J) \vee (1 = 1))) \wedge \\ (L \Rightarrow ((K \vee J) \vee (0 = 1))) \wedge \\ (M \Rightarrow (\neg K \vee (0 = 0))) \wedge \\ (M \Rightarrow ((K \vee \neg J) \vee (1 = 0))) \wedge \\ (M \Rightarrow ((K \vee J) \vee (0 = 0)))\end{gathered}$$

Note that until this very step equivalence to the original ϕ is being kept.

Since ϕ has changed we can repeat steps (1)–(4). Namely, the step (2) may transform the following disjunct of ϕ':

$$S \equiv (M \Rightarrow ((K \vee \neg J) \vee (1 = 0))).$$

The relation graph has been produced for S. Consider the following component of the graph:



The label X stands for $(cnt\ 0\ (-\ n\ 1)\ key\ a)$ whereas Y means $entr$. Remind that this graph component is actually formula E.

Which subgoal will lead to modification of ϕ'? In fact it is $\neg J$ corresponding to the pattern $g(c, d)$ where g is "$\neq$", c is $entr$ and d is $(rep1\ n\ key\ entr\ a)$. So, the searching procedure begins to look for a function call corresponding to the variable $entr$. The search begins at node X. During the search a subgraph of the relation graph emerges. This subgraph is exactly the component demonstrated above. The expression $(cnt\ 0\ (-\ n\ 1)\ key\ a)$ is the function call we were looking for. The conjunct E is the path we need. So, the expression $(cnt\ 0\ (-\ n\ 1)\ key\ a)$ must be assigned to variable v, and E becomes the value of q.

As a result we have the new formula

$$T \equiv (=\ (cnt\ 0\ (-\ n\ 1)\ key\ a)\ (rep1\ n\ key\ entr\ a)).$$

Let Z denote the disjunct S after replacement of the goal $\neg J$ by T:

$$(M \Rightarrow ((K \vee T) \vee (1 = 0))).$$

Note that $Z \not\sim S$ but the truth of S follows from the truth of Z. So we may replace S by Z in ϕ' which results in formula ϕ'':

$$\begin{gathered}(L \Rightarrow (\neg K \vee (0 = 1)))\wedge \\ (L \Rightarrow ((K \vee \neg J) \vee (1 = 1)))\wedge \\ (L \Rightarrow ((K \vee J) \vee (0 = 1)))\wedge \\ (M \Rightarrow (\neg K \vee (0 = 0)))\wedge \\ (M \Rightarrow ((K \vee T) \vee (1 = 0)))\wedge \\ (M \Rightarrow ((K \vee J) \vee (0 = 0)))\end{gathered}$$

And here it is, the result of our strategy. On the step (6) ACL2 is able to prove ϕ'' by induction on n thus validating the original VC ϕ.

5 Conclusion

In this paper we briefly described a new approach to verification of Sisal programs iterating over data structures. The approach owes its success to integration of three different projects. First, question of the Sisal program correctness

is reduced to correctness of a corresponding C program. Second, our C-light verification system is able to handle it. Finally, we can facilitate verification of restricted loop cases by means of a symbolic method.

As a result of such integrating experiments we developed a proof strategy for loops. This heuristic approach involves such formula rewritings that correctness of resulting formula provides correctness of the original one. However, the total equivalence may be lost during application of a procedure related to equality/inequality. The study case in this paper illustrates successful application of the strategy.

In general case our strategy does not guaranty success, even for a true verification condition. Perhaps, it will require some revisions based on the main principle: the truth of resulting formula implies validity of the original one.

Our future experiments will include less artificial study cases. The first goal is verification of Sisal programs for sorting and linear algebra. Such programs imply iterations over vectors and matrices, thus making them appropriate objects for the symbolic verification method. A more distant task consists of developing of axiomatics semantics for Sisal.

We also keep in mind possibility to use the C language as a mediator for other languages, besides Sisal. The language of computable predicates `P` [26] is a good candidate. This is a functional language aimed to verification. Our colleagues are already developing a translator into the C language.

Acknowledgment. The authors are grateful to Prof. Nikolay Shilov (Univ. of Innopolis, Kazan, Russia) for critical analysis of our proof strategy.

References

1. Apt K. R., Olderog E. R. Verification of sequential and concurrent programs. Berlin etc. Springer, 1991. 450 p.
2. Ball T., Hackett B., Lahiri S. K., Qadeer S., Vanegue J. Towards Scalable Modular Checking of User-Defined Properties. LNCS, 2010, vol. 6217, pp. 1–24.
3. Barnett M., Leino K. R. M., Schulte W., The Spec# Programming System: An Overview. LNCS, 2005, vol. 3362, pp. 49-69.
4. Cann D. C. The Optimizing Sisal Compiler. Livermore, CA, 1992. 74 p. (Tech. Rep. / Lawrence Livermore National Laboratory; UCRL-MA-110080)
5. Cohen E., Dahlweid M., Hillebrand M. A., Leinenbach D., Moskal M., Santen T., Schulte W., Tobies S. VCC: A Practical System for Verifying Concurrent C. LNCS, 2009, vol. 5674, pp. 23-42.
6. Filliâtre J. C., Marché C. Multi-prover verification of C programs. LNCS, 2004, vol. 3308, pp. 15-29.
7. Filliâtre J. C., Marché C. The Why/Krakatoa/Caduceus Platform for Deductive Program Verification. LNCS, 2007, vol. 4590, pp. 173-177.
8. Filliâtre J. C., Paskevich A. Why3 — Where Programs Meet Provers. LNCS, 2013, vol. 7792, pp. 125-128.
9. Gluhankov M. P., Dortman P. A., Pavlov A. A., Stasenko A. P. Translating components of a functional programming system (SFP). Modern problems of program construction. — Novosibirsk, 2002. — P. 69-87, URL: https://www.iis.nsk.su/files/articles/sbor_kas_09_gluhankov_etc.pdf (In Russian)

10. Heras J., Komendantskaya E., Johansson M., Maclean E. Proof-Pattern Recognition and Lemma Discovery in ACL2. LNCS, 2013, vol. 8312. pp. 389–406.
11. Kasyanov V. Sisal 3.2: functional language for scientific parallel programming. Enterprise Information Systems, 2013, 7(2), pp. 227–236.
12. Kasyanov V., Kasyanova E. A System of Functional Programming for Supporting of Cloud Supercomputing. WSEAS Transactions on Information Science and Applications, 2018, 15(9), pp. 81–90.
13. Kasyanov V. N., Kasyanova E. V. Methods and system of cloud parallel programming. Proceedings of the XIV International Asian School-Seminar on Problems of Optimizing Complex Systems, Part 1. Almaty, Kazakhstan, 2018. P. 298-307 (In Russian)
14. Kasyanov V. N., Kasyanova E. V. Programming language Cloud Sisal. Novosibirsk, Russia. A. P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences. Preprint N181. 2018. 45 p. URL: https://www.iis.nsk.su/files/preprints/Preprint-kasyanovy.pdf (In Russian)
15. Kaufmann M., Moore J. S. An Industrial Strength Theorem Prover for a Logic Based on Common Lisp. IEEE Transactions on Software Engineering, 1997, 23(4), pp. 203–213.
16. Kondratyev D. Implementing the Symbolic Method of Verification in the C-Light Project. LNCS, 2018, vol. 10742, pp. 227–240.
17. Kondratyev D. Automated Sisal program verification with ACL2. Appendices. URL: https://bitbucket.org/Kondratyev/verify-sisal
18. Leino K. R. M. Program proving using intermediate verification languages (IVLs) like Boogie and Why3. Proceedings of the 2012 ACM conference on High integrity language technology. N.Y.: Association for Computing Machinery, 2012. pp. 25–26.
19. Leino K. R. M., Rümmer P. A Polymorphic Intermediate Verification Language: Design and Logical Encoding. Lecture Notes in Computer Science. Berlin: Springer-Verlag, 2010. Vol. 6015. P. 312-327.
20. Li J., Sun J., Li L., Loc Le Q., Lin S-W. Automatic loop-invariant generation and refinement through selective sampling. Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, 2017. pp. 782–792.
21. Mandrykin M. U., Khoroshilov A. V. High-level memory model with low-level pointer cast support for Jessie intermediate language. Programming and Computer Software, 2015, 41(4), pp. 197–207.
22. Maryasov I. V., Nepomniaschy V. A., Kondratyev D. A. Invariant Elimination of Definite Iterations over Arrays in C Programs Verification. Modeling and Analysis of Information Systems, 2017, 24(6), p. 743–754.
23. Maryasov I. V., Nepomniaschy V. A., Promsky A. V., Kondratyev D. A. Automatic C Program Verification Based on Mixed Axiomatic Semantics. Automatic Control and Computer Sciences, 2014, 48(7), pp. 407–414.
24. Nepomniaschy V. A. Symbolic Method of Verification of Definite Iterations over Altered Data Structures. Programming and Computer Software, 2005, 31(1), pp. 1–9.
25. Nepomniaschy V. A., Anureev I. S., Promskii A. V. Towards Verification of C Programs: Axiomatic Semantics of the C-kernel Languages. Programming and Computer Software, 2003, 29(6), pp. 338–350.
26. Shelekhov V. I., Tumurov E. G. Applying Automata-based Software Engineering for the Lift Control Program. Programmnaya Ingeneria, 2017, 8(3), pp. 99–111. URL: http://persons.iis.nsk.su/files/persons/pages/lift1.pdf (In Russian)

27. Tschannen J., Furia C. A., Nordio M., Meyer B. Verifying Eiffel programs with Boogie. Proceedings of the First International Workshop on Intermediate Verification Languages, BOOGIE (2011). Wroclaw, Poland, August 2011, URL: http://arxiv.org/abs/1106.4700
28. Tuerk T. Local Reasoning about While-Loops. VSTTE 2010 Workshop Proceedings, 2010, pp. 29–39. URL: https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/68924/eth-5084-01.pdf
29. Ulbrich M. Dynamic Logic for an Intermediate Language: Verification, Interaction and Refinement. Berlin, epubli GmbH, 2014, 268 p.

The Analytical Object Model as a Base of Heterogeneous Data Integration

Korobko Anna[1] and Metus Anna[1]

Institute of Computational Modeling of the SB RAS, Krasnoyarsk, Russia
lynx@icm.krasn.ru

Abstract. When viewed the issue of analytical integration of heterogeneous data without warehouse building the unified model of diverse data sources has to be suggested. The desired model has to take into account analytical features of original file formats, to provide a construction of the integral analytical model and to attend to unlimited user data queries. This paper proposes the analytical object model in terms of a formal specification as the unified model and presents mapping of a XSD schema and a relational database to this model. The model has been applied to analyze The All-Russia website of procurement that uses XML and The Local System of procurement that uses relation DB. The model instances obtained for each format are partly represented in this paper in the form of JSON.

Keywords: OLAP· integral analytic model· analytical integration of heterogeneous data· MDA· XML

1 Introduction

One of the most important aspects of the evolution of On-Line Analytical Processing (OLAP) is developing theoretical approaches to simultaneous analysis of heterogeneous data. OLAP tools are extensively used in decision support systems assisting managers of large companies with advanced analysis and reporting. As usually, a lot of business information flows from internal sources, that provides an accumulation of the great amount of operational data [1]. In some sources [2], the internal data that are owned by the decision maker and can be directly incorporated into the decisional process are called stationary. Each source is a special-purpose data store associated with its own data format. Joined internal sources represent a model for heterogeneous data. However, with significant growth of the number of open access databases, it becomes possible to involve external data in decision-making process for extra benefits. Valuable external data which may be related, for instance, to the market, to competitors, or to potential customers, are called situational data [3]. Well-informed and effective decisions often require a tight relationship between stationary and situational data [2].

Analysis of municipal procurement is the task demanding simultaneous analysis of heterogeneous data. According to The Federal Law N 44-FZ "Contract

system in the procurement of goods, works and services for state and municipal needs" the Official All-Russia website of procurement (zakupki.gov.ru) has been developed. It consolidates municipal demands, ongoing purchases and contracts all over the country. It sends and receives data in XML format according to system pre-defined XSD schemas. Otherwise, the Local System of procurement forms municipal demand orders and scheduled plan of purchases. The Local System has a bidirectional link with the All-Russia website of procurement. Also, the Local System uses Oracle DBMS to store data and metadata. From the regional government perspective the Local System data is an internal data source and the All-Russia website data is an external one. An analyst, who wants to trace some purchase from a demand order to a contract, or to analyze some supplier activity, needs to integrate these heterogeneous data.

Complex analysis of internal and external data together concerns reconciling (merging) diverse data sources. Often this step is performed by data warehouse building or integration of separate analysis results while representing. This process requires highly qualified analysts to intent and extra time for data actualization and preparation.

We can consider related works from different points of view. Nowadays, the complex analysis of heterogeneous data is actively discussed by researchers [4–11]. In speaking about ideology, there are several conceptual foundations for OLAP technology development which have been suggested by modern leading researchers. Thus, the concept of self-service Business Intelligence [2, 12] reduces the requirements for user skills. On the other hand, the concept of exploratory OLAP supports ad-hoc arbitrary query execution [1, 13]. The conceptual description of these approaches shows further researches a way forward, leaving them a wide discretion for realization. From the logical viewpoint, modern researches unanimously recognize the need for constructing a global (or a mediated) schema that enables uniform access to the data [14]. The particular realizations of the approaches are rather different. They fall somewhere on the spectrum between warehousing and virtual integration [15]. Approaches to extraction, transformation and loading data to a centralized warehouse are proposed in [16, 17]. But it requires highly experienced modelers and designers to compare a wide variety of domain concepts. Standards of heterogeneous data interchange have been already developed that provide creating an unified format of data exchange [18]. However, there arent common algorithms for transforming miscellaneous data into this specific standard. As a virtual global schema, ontology is proposed [14] for information disclosure form integrated data. This approach isnt aimed for analytical processing notwithstanding its grace and feasibility. The analytical model of data source [19] has been suggested under the virtual schema approach. Moreover, it allows us to reduce the requirements to user-analysts skills so that analytical concepts (measures and dimensions) can be arranged according to their analytical features that mean grouped into the same request allowed by data consistency. The model serves to view all available data in a multidimensional form and provides unlimited querying without knowledge specified about database structure, functional dependencies and SQL. Another

technology concept of operational analysis of heterogeneous data was suggested to avoid warehouse building. This technology constructs the integral analytical model by comparing and integrating of original data sources automatically. This integral model supports the design and performance of random user data query straight to an original data store and delineates author vision of implementation of heterogeneous data analysis.

Analytical integration of heterogeneous data without warehouse building requires the unified model of diverse data sources has to be suggested. The desired model has to take into account the analytical features of original file formats, to provide a construction of the integral analytical model and to attend to unlimited user data queries.

This paper proposes the analytical object model (AOM) as the unified model. The AOM is a metamodel describing the structure for model instances of original data sources regardless of its format. Mapping of a XSD schema and a relational database to this model is presented. Matching of source structure items for model items in municipal procurement analysis is presented.

2 The formal specification of the analytical object model

Development of the analytical object model accords with the model-driven development (MDD) of information systems, based on 4 level modeling [18]. The highest level (M3) describes a modeling specification and the lowest one (M0) describes program system data. To develop the analytical object model model-driven approach allows creating metamodel (M2) for describing the model instance structure of the original data source regardless of its format. During system lifecycle this model services to sources metadata store, sources link details and instances production and maintain order. At the abstract layer the analytical object model is a base of heterogeneous system integration and unifies data format particularities.

Furthermore, the analytical object model has to take into account the analytical features of original file formats to provide a multidimensional form for OLAP. Consortium OMG specifies open standard Common Warehouse Metamodel to combine multidimensional modeling and model-driven approach for data warehouses. The standard consists of the set of metamodels for data interchange within the conceptual layer and assumes data moving to a warehouse manually. Model instances production taking into account implementation requirements are beyond the standard scope, so it needs to be produced additionally. Automatical source integration without warehouse building requires the standard extension by adding analytical object metamodel. Suggested metamodel determinates the unified structure of a source regardless of its format and consists of analytical classes such as "AssociationClass", "DAttribute", "FAttribute" "Hierarchy". "AssociationClass" is a container class for other items. It matches the source items of a top level. The model has been applied to analyze municipal procurement. There are XML and relational data formats in this task.

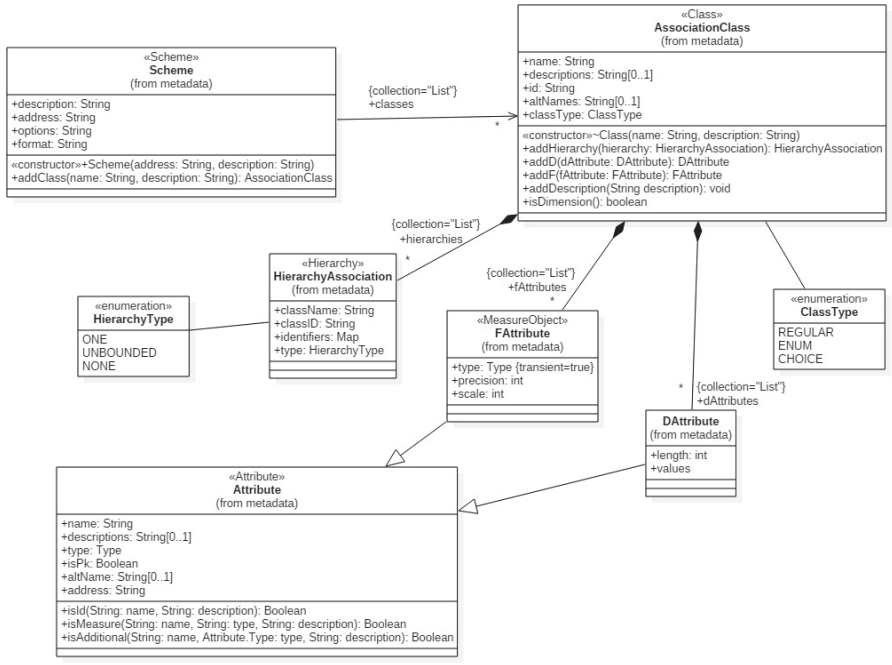


Fig. 1. A class diagram for the analytical object model for integrating sources.

So, integration of these two sources model instances is produced from XML Schema Definition and Oracle database. The class diagram of the analytical object model is shown in Figure 1 using UML.

"AssociationClass" instances are created for each table in relation source and for each complex type in XSD. Inner simple types and relation table column relates "DAttribute" class (descriptive attribute) or "FAttribute" class (fact attribute) depending on its analytical features and links "AssociationClass" with the composite association. "Hierarchy" class instances describe analytical relations between source items, it matches foreign keys in a relational data source and parent-child relations in XSD. Full rules to produce analytical model instances for relation DB and XSD metadata are shown in Table 1.

The unified representation of diverse data sources in the form of the analytical object model allows producing a single algorithm of multidimensional form constructing and merging structures of heterogeneous data sources into an integral analytical model. Like traditional multidimensional approach numerical data produces measures, descriptive attribute forms dimensions and "Hierarchy" instances arrange dimensions hierarchically. Every model class has features to facilitate both multidimensional modeling and user query support. "Scheme" class has the connection properties of physical data sources.

Table 1: Matching of XSD and relation DB metadata and analytical object model items

Begin of Table1		
Analytical object model	**Relation DB metadata**	**XSD metadata**
class AssociationClass	a table	- a complexType - a simpleType (enumeration restricted) - a choice. *elements with the same name (type, if exist) and inner elements relate the same AssociationClass
+name: String	a table name	value of a element name attribute
+descriptions: String [0..*]	- a table description or a table name in Russian; - a description of foreign keys to this table	value of a documentation element
+id	an auto increment identifier	
+altNames: String [0..*]	a foreign key name to this table	value of a complexType name attribute
+classType: [REGULAR, ENUM, CHOICE]	REGULAR (fixed value)	REGULAR - if an element type is a complexType; ENUM - if an element is a restricted by enumeration one; CHOICE - if it is a choice element.
class DAttribute	a column, if it's of string, boolean or primary key types	an element based on a simpleType (String, Boolean Integer type)
+name: String	a column name value of	an element name attribute
+descriptions: String [0..*]	a column description	- value of a documentation element; - value of a simpleType name attribute
+type: [BOOLEAN, DATE, DATETIME, INTEGER, NUMERIC, STRING]	system data types	
+isPK: Boolean	TRUE if the column contains a primary key, otherwise FALSE	TRUE if an element is required, otherwise FALSE

Continuation of Table 1		
Analytical object model	**Relation DB metadata**	**XSD metadata**
+altName	no value	value of an simpleType name attribute if a type of the element relates the one
+values	no value	enumeration values for restricted simpleType
+address	no value	a specified wildcard (@) - if an element is an attribute, otherwise - no value
+length	a data type length	a data type length, according to restrictions: - facets maxLength, maxExclusive-1, maxInclusive; - pattern value. - 0 unrestricted length.
class "FAttribute"	a column, if it's of decimal type	an element based on a simpleType (Double, Decimal)
+name: String	a column name	value of an element name attribute
+descriptions: String [0..*]	a column description	- value of a documentation element; - value of a simpleType name attribute
+type	system data types	NUMERIC fixed for xs:double, xs:decimal
+isPK	FALSE fixed	FALSE fixed
+altName	no value	value of an simpleType name attribute if a type of the element relates the one
+address	no value	a specified wildcard (@) - if an element is an attribute, otherwise - no value
+precision	count of numbers	according to facets totalDigits, pattern. otherwise 0 unrestricted.
+scale	count of digits following the decimal point	according to facets ractionDigits, pattern. otherwise 0 unrestricted.

Continuation of Table 1		
Analytical object model	**Relation DB metadata**	**XSD metadata**
class "HierarchyAssociation"	a foreign key	parent-child relation
+className	a name of the table related by the foreign key	a name of an inner complexType element
+classID	an unique id of the table related by the foreign key	an unique id of the inner complexType element related by the foreign key
+identifiers	pairs of foreign key - primary key	no value
+type: [ONE, UNBOUNDED]	ONE - fixed	ONE - if maxOccures attribute value of the child element equals one UNBOUNDED - if maxOccures attribute value of the child element more than one

Having produced the AOM, an analyst is able to select objects for analysis regardless of which source owns them. After the objects are selected, the AOM maintains the query construction process due to preserving structure peculiarities of the data source format.

3 An example of analytical object model instance producing

According to the approach a program system was developed to produce an instance of an analytical object model for a source to be integrated automatically. The program system has services specified for XML and relational data format. Each service is capable to construct AOM for one of the format types and to send it to a server in the form of JSON. The server is capable to construct a multidimensional model and subsequently to produce the integral analytical model. The AOM instance obtained for XSD format when analyzed the All-Russia website of procurement (zakupki.gov.ru) is partly shown in Figure 2 in the form of JSON.

The "Hierarchies" elements of "product" class accord with the relation between "Hierarchy" and "AssociationClass". Particularly according to this instance a product relates the All-Russian Classifier of Products (OKPD) and the All-Russia Classifier of Measurement Units (OKEI). The analytical object model takes into account both XML peculiarities and relational database ones. Another AOM instance obtained for relation format when analyzed the Local System of procurement is partly shown in Figure 3 in the form of JSON. This part of the

```
{ ▼ 5 properties, 366 bytes
  "name": "OKEI",
  "description": "Reference to OKEI",
  "hierarchies": [],
  "dAttributes": [ ▼ 3 items, 260 bytes
    { ▶ 5 properties, 84 bytes "name": "code", "description": "Cod", "ty…},
    { ▶ 4 properties, 89 bytes "name": "nationalCode", "description": "N…},
    { ▶ 4 properties, 79 bytes "name": "fullName", "description": "Full …}
  ],
  "fAttributes": []
},
{ ▼ 5 properties, 1015 bytes
  "name": "product",
  "description": "The object of the procurement",
  "hierarchies": [ ▼ 2 items, 117 bytes
    { ▶ 2 properties, 58 bytes "className": "OKPD", "identifiers": { 1 p…},
    { ▼ 2 properties, 55 bytes
      "className": "OKEI",
      "identifiers": { ▼ 1 property, 20 bytes
        "OKEI_code": "code"
      }
    }
  ],
  "dAttributes": [ ▼ 3 items, 284 bytes
    { ▶ 4 properties, 84 bytes "name": "sid", "description": "Unique ide…},
    { ▶ 4 properties, 108 bytes "name": "externalSid", "description": "Ex…},
    { ▶ 4 properties, 88 bytes "name": "name", "description": "Product n…}
  ],
  "fAttributes": [ ▼ 5 items, 502 bytes
    { ▼ 4 properties, 93 bytes
      "name": "price",
      "description": "Unit price in the contract currency",
      "precision": 10,
      "scale": 0
    },
    { ▶ 4 properties, 91 bytes "name": "priceRUR", "description": "Unit …},
    { ▶ 4 properties, 81 bytes "name": "sum", "description": "Cost in co…},
    { ▶ 4 properties, 83 bytes "name": "sumRUR", "description": "Cost in…},
    { ▶ 4 properties, 142 bytes "name": "quantity", "description": "Quant…}
  ]
}
```


Fig. 2. A part of the analytical object model instance for fcsExtegration.xsd

AOM instance presents the All-Russia Classifier of Measurement Units (OKEI) and its descriptive attributes.


```
{ ▼ 5 properties, 951 bytes
  "name": "FOKEI",
  "description": "Russian classifier of measurement units",
  "hierarchies": [],
  "dAttributes": [ ▼ 9 items, 822 bytes
    { ▶ 4 properties, 78 bytes "name": "FOKEI", "description": "Name title… },
    { ▶ 4 properties, 89 bytes "name": "RUSSIAN", "description": "Symbol c… },
    { ▶ 4 properties, 94 bytes "name": "ENGLISH", "description": "Symbol c… },
    { ▶ 4 properties, 82 bytes "name": "COD_3", "description": "Three-digi… },
    { ▶ 5 properties, 94 bytes "name": "IDFOKEI", "description": "Identifi… },
    { ▶ 4 properties, 89 bytes "name": "SECT", "description": "Classifier … },
    { ▶ 4 properties, 95 bytes "name": "GRUPA", "description": "Code group… },
    { ▶ 4 properties, 74 bytes "name": "COD", "description": "OKEI code", … },
    { ▶ 3 properties, 107 bytes "name": "RU", "description": "The basis of … }
  ],
  "fAttributes": []
}
```


Fig. 3. A part of the analytical object model instance for the Local System of procurement

The popularity of relational and XML data formats allows involving additional information concerning the business environment from a large number of open data sources in the analysis. So, XML format is used by Federal State Statistics Service for social and macroeconomics statistics [19], by the Central Bank of Russia for financial market indicators [20], by Federal Tax Service for open governmental data [21]. Also, a large number of research and academic institutions across the world create relational databases in various fields of science and technology, which are available free through web portals [22].

Complete deployment of the program system based on the suggested model and its successful beta testing on analyzing municipal procurement data verify the approach. Further developing of the technology of integral analytical modeling concern producing and testing of the algorithm of multidimensional view forming based on AOM.

4 Conclusion

The analytical object model has been suggested. The model formally describes the analytical and structural peculiarities of heterogeneous data sources to overcome their diversity and to allow them to be integrated automatically. Possibility to include some source to the integral model without warehouse building is provided with analyzing of analytical features and relations of an original source format. Retaining a format metadata arrangement contributes to supporting unlimited user data queries.

References

1. Ibragimov, D., Hose, K., Pedersen, T.B., Zimnyi, E.: Towards Exploratory OLAP over Linked Open DataA Case Study. Enabling Real-Time Bus. Intell. 118 (2014).
2. Abell, A., Darmont, J., Etcheverry, L., Golfarelli, M., Mazn, J.-N., Naumann, F., Pedersen, T., Rizzi, S.B., Trujillo, J., Vassiliadis, P., Vossen, G.: Fusion Cubes: towards self-service Business Intelligence. Int. J. Data Warehous. Min. 9, 6688 (2013).
3. Lser, A., Hueske, F., Markl, V.: Situational business intelligence. In: Lecture Notes in Business Information Processing (2009).
4. Gallinucci, E., Golfarelli, M., Rizzi, S., Abell, A., Romero, O.: Interactive multidimensional modeling of linked data for exploratory OLAP. Inf. Syst. 77, 86104 (2018).
5. Alpar, P., Schulz, M.: Self-Service Business Intelligence. Bus. Inf. Syst. Eng. 58, 151155 (2016).
6. Chen, H., Storey, V.C., Chen, Hsinchun and Chiang, Roger HL and Storey, V.C.: Business Intelligence and Analytics: From Big Data To Big Impact. Mis Q. 36, 11651188 (2012).
7. Singh, R., Yoon, V.Y., Redmond, R.T.: Integrating Data Mining and On-line Analytical Processing for Intelligent Decision Systems. J. Decis. Syst. 11, 185204 (2002).
8. Baranovi, M., Kalpi, D., Brki, L.: Application of Semantic and Structural Similarity for Schema Reuse in Conceptual Database Design. Proc. 6th WSEAS Eur. Comput. Conf. (ECC 2012). 368373 (2012).
9. Cuzzocrea, A., Bellatreche, L., Song, I.-Y.: Data warehousing and OLAP over big data: current challenges and future research directions. In: Proceedings of the sixteenth international workshop on Data warehousing and OLAP - DOLAP 13. pp. 6770 (2013).
10. Pe, J.M., Rafael, B., Aramburu, M.J., Pederson, T.B.: Integrating Data Warehouses with Web Data: A Survey. IEEE Trans. Knowl. Data Eng. 20, 940955 (2008).
11. Salem, R., Boussad, O., Darmont, J.: Active XML-based Web data integration. Inf. Syst. Front. 15, 371398 (2013).
12. Varga, J., Romero, O., Pedersen, T.B., Thomsen, C.: Analytical metadata modeling for next generation BI systems. J. Syst. Softw. 144, 240254 (2018).
13. Rizzi, S., Gallinucci, E., Golfarelli, M., Romero, O., Abell, A.: Towards Exploratory OLAP on Linked Data. In: 24th Italian Symposium on Advanced Database Systems, SEBD 2016. pp. 8693 (2016).
14. Benedikt, M., Cuenca Grau, B., Kostylev, E. V.: Logical foundations of information disclosure in ontology-based data integration. Artif. Intell. 262, 5295 (2018).
15. AnHai, D., Alon, H., Zachary, I.: Principles of Data Integration. Elsevier (2012).
16. Lujn-Mora, S., Vassiliadis, P., Trujillo, J., Lujan-Mora, S., Vassiliadis, P., Trujillo, J.: Data mapping diagrams for data warehouse design with UML. Concept. Model. 2004. 191204 (2004).
17. Kimball, R., Ross, M.: The Data Warehouse Toolkit, The Definitive Guide to Dimensional Modeling. (2013).
18. Omg, Object Management Group: Object Management Group, Model Driven Architecture (MDA). OMG Doc. ormsc/2014-06-01. 2.0, 115 (2014).
19. Korobko, A.V., Penkova, T.G.: On-line analytical processing based on formal concept analysis. In: Procedia Computer Science (2010).
20. Federal State Statistics Service, http://www.gks.ru/wps/wcm/connect/rosstat_main/rosstat/ru/statistics/accounts/.

21. Central Bank of Russia for financial market indicators, http://www.cbr.ru/development/DWS/.
22. Federal Tax Service for open governmental data, https://www.nalog.ru/opendata/.
23. Listing of Open Access Databases - LOADB, http://www.loadb.org/Control.do?_brse.

Computable Topology for Reliable Computations *

Margarita Korovina[1] and Oleg Kudinov[2]

[1] A.P. Ershov Institute of Informatics Systems, SbRAS,
rita.korovina@gmail.com
[2] Sobolev Institute of Mathematics, SbRAS, Novosibirsk State University,
kud@math.nsc.ru

Abstract. Using the framework of computable topology we investigate computable minimality of lifted domain presentations of computable Polish spaces, in particular the real numbers, the Cantor and Baire spaces, the continuous functions on a compact interval, which are widely used in theoretical computer science, e.g., automata theory, computable analysis and reliable computations. We prove that all lifted domain presentations for computable Polish spaces are computably and topologically minimal. We establish a criterion of computable minimality for any effective domain presentation. Then we show that a naive adaptation of the notion of stability from computable model theory does not work. Instead of that we propose another approach based on principal translators and prove that in the case of the real numbers we can effectively construct a principal computable translator from the lifted domain presentation to any other domain presentation.

Keywords: computable topology, reliable computation, computable analysis, lifted domain.

1 Introduction

Computations over continuous data are central in scientific computing and engineering. This motivates research in investigating properties of different frameworks for representing computable objects and computations over them. One such frameworks is the well established domain theory approach proposed by D. Scott [23] and Yu.L. Ershov [3], where computational processes and a data types are modelled using appropriate algebraic or continuous domains.

Informally, any representation of a space X is the factorisation of a part $\widetilde{\mathbb{D}}$ of an appropriate algebraic domain $\mathbb{D}$ up to homeomorphism. The existence and uniqueness of representations of spaces by (algebraic) Scott-Ershov domains have been investigated in [4, 5], where the some canonical representations have been introduced. However the uniform characterisations of such representations

* The research leading to these results has received funding from the DFG grants WERA MU 1801/5-1 and CAVER BE 1267/14-1 and RFBR grant A-17-01-00247.

have not been proposed in details. In this paper, we address similar problems in the setting of continuous domains which are more suitable, in practice, for continuous data computations [23, 2]. We aimed at uniting independently developed concepts of computability on computable metric spaces and on weakly effective $\omega-$continuous domains [26, 11, 17].

In particular, following ideas from [28, 1, 13], we propose homeomorphic embeddings of computable Polish spaces into the lifted domains that endow the original spaces with computational structure and investigate the following natural problems. One of them whether lifted domain presentations of computable Polish spaces are computably and/or topologically minimal. Another one concerns a characterisation of translators from lifted domain presentations to other domain presentations of computable Polish spaces. These problems originated in computable model theory [19, 21] and are related to stability of structures and spaces. In particular, we show the cases when computable and continuous translations between different presentations of spaces exist and/or unique.

In this paper we introduce a key notion of computably minimal domain presentation and show that canonical lifted domain presentations are computably minimal. Moreover, since there are infinitely many computable translators from any computably minimal domain presentation to any computable one (see, Section 3.4), using the idea of a principal (maximal) computable numbering from recursion theory [24] we introduce the notion of a principal computable (continuous) translator and prove that in the case of the real numbers we can effectively construct a principal computable translator from the lifted domain presentation to any other domain presentation.

The paper is organised as follows. In Section 2 we give some basic concepts from domain theory, computable Polish spaces and effectively enumerable topological spaces. We use effectively enumerable T_0-spaces as a uniform framework to represent computability on domains and computable Polish spaces. Section 3 contains the main contributions of the paper. We first computably embed a computable Polish space $\mathbb{P}$ into a lifted domain $\mathbb{D}^{\mathbb{P}}$ that is weakly effective ω–continuous domain such that the Scott topology on $\mathbb{D}^{\mathbb{P}}$ agrees with the standard topology on $\mathbb{P}$ and the proposed embedding is a homeomorphism between the set of maximal elements and the original space. We prove that lifted domain presentations are computably and topologically minimal. We establish a criterion of computable mininality for any effective domain presentation. Then we show that while all topologically minimal presentations of computable Polish spaces are not stable there exist principal translators in the case of the lifted domain presentation of the reals.

2 Preliminaries

We refer the reader to [11, 13, 14, 26, 25, 27] for basic definitions and fundamental concepts of computable topology and to [2, 1] for basic definitions and fundamental concepts of domain representations of reliable computations. A numbering of a set Y is a total surjective map $\gamma : \omega \to Y$. We use the standard denota-

tions for the real numbers $\mathbb{R}$ and $C([a,b])$ for the set of continuous real-valued functions defined on a compact interval $[a,b]$.

2.1 Weakly effective ω-continuous domains

In this section we present a background on domain theory. The reader can find more details in [2, 10]. Let $(D;\perp,\leq)$ be a partial order with a least element $\perp$. A subset $A \subseteq D$ is *directed* if $A \neq \emptyset$ and $(\forall x,\, y \in A)(\exists z \in A)(x \leq z \wedge y \leq z)$. We say that D is a *directed complete partial order*, denoted dcpo, if any directed set $A \subseteq D$ has a supremum in D, denoted $\bigsqcup A$. For two elements $x,\, y \in D$ we say x is *way-below* y, denoted $x \ll y$, if whenever $y \leq \bigsqcup A$ for a directed set A, then there exists $a \in A$ such that $x \leq a$. A function $f : \mathbb{D} \to \widetilde{\mathbb{D}}$ between cpos is continuous if f is monotone and for each directed set $A \subseteq D$ we have $F(\bigsqcup A) = \bigsqcup\{f(x) \mid x \in A\}$. We say that $\mathbf{B} \subseteq D$ is a *basis* (base) for D if for every $x \in D$ the set $approx_{\mathbf{B}}(x) = \{y \in \mathbf{B} \mid y \ll x\}$ is directed and $x = \bigsqcup approx_{\mathbf{B}}(x)$. We say that D is *continuous* if it has a basis; it is *ω–continuous* if it has a countable basis.

Definition 1. *[10] Let $\mathbb{D} = (D;\mathbf{B},\beta,\leq,\perp)$ be an ω–continuous domain with a basis $\mathbf{B}$ and its numbering $\beta : \omega \to B$. We say that $\mathbb{D}$ is weakly effective if the relation $\beta(i) \ll \beta(j)$ is computably enumerable.*

One of the well-known examples of weakly effective ω–continuous domains is the interval domain $\mathcal{I}_{\mathbb{R}} = \{[a,b] \mid a,\, b \in \mathbb{R},\ a \leq b\} \cup \perp$ with the inverse order and the countable basis that is the collection of all compact intervals with rational endpoints together with the least element $\perp$ (see e.g. [2]).

Proposition 1 (Interpolation Property). *[10] Let D be a continuous domain and let $M \subseteq D$ be a finite set that $(\forall a \in M)\, a \ll y$. Then there exists $x \in D$such that $M \ll x \ll y$ holds. If B is a basis for D then x may be chosen from B.*

2.2 Perfect Computable Polish spaces

In this paper we work with the following notion of a computable Polish space abbreviated as CPS. A *perfect computable Polish space*, simply *computable Polish space*, is a complete separable metric space $\mathbb{P}$ without isolated points and with a metric $d : P \times P \to \mathbb{R}$ such that there is a countable dense subset $\mathcal{B}$ called a *basis of* $\mathbb{P}$ with the numbering $\alpha : \omega \to \mathcal{B}$ that makes the following two relations: $\{(n,m,i) \mid d(\alpha(n),\alpha(m)) < q_i,\ q_i \in \mathbb{Q}\}$ and $\{(n,m,i) \mid d(\alpha(n),\alpha(m)) > q_i,\ q_i \in \mathbb{Q}\}$ computably enumerable (c.f. [28]).

The standard notations $B(x,y)$ and $\overline{B}(x,y)$ are used for open and closed balls with the center x and the radius y. We also use the notation $\alpha(n) = b_n$ for a numbering $\alpha : \omega \to \mathcal{B}$.

2.3 Effectively Enumerable Topological Spaces

Let (X, τ, α) be a topological space, where X is a non-empty set, $B_\tau \subseteq 2^X$ is a base of the topology τ and $\alpha : \omega \to B_\tau$ is a numbering. In notations we skip τ since it can be recovered by α. Further on we will often abbreviate (X, α) by $\mathbb{X}$ if α is clear from a context.

Definition 2. *[17] A topological space (X, α) is effectively enumerable if there exists a computable function $g : \omega \times \omega \times \omega \to \omega$ such that $\alpha(i) \bigcap \alpha(j) = \bigcup_{n \in \omega} \alpha(g(i, j, n))$. and the set $\{i \mid \alpha(i) \neq \emptyset\}$ is computably enumerable.*

Definition 3. *Let (X, α) be an effectively enumerable topological space.*

1. *A set $\mathcal{O} \subseteq X$ is effectively open if there exists a computably enumerable set V such that $\mathcal{O} = \bigcup_{n \in V} \alpha(n)$.*
2. *A sequence $\{\mathcal{O}_n\}_{n \in \omega}$ of effectively open sets is called computable if there exists a computable sequence $\{V_n\}_{n \in \omega}$ of computably enumerable sets such that $\mathcal{O}_n = \bigcup_{k \in V_n} \alpha(k)$.*

Let $\mathcal{O}_X$ denote the set of all open subsets of X and $\mathcal{O}^e_X$ denote the set of all effectively open subsets of X.

Definition 4. *[14] Let $\mathbb{X} = (X, \alpha)$ be an effectively enumerable topological space and $\mathbb{Y} = (Y, \beta)$ be an effectively enumerable T_0–space. A function $f : \mathbb{X} \to \mathbb{Y}$ is called* partial computable *(pcf) if the following properties hold. There exist a computable sequence of effectively open sets $\{O_n\}_{n \in \omega}$ and a computable function $H : \omega^2 \to \omega$ such that*

1. $\mathrm{dom}(f) = \bigcap_{n \in \omega} O_n$ *and*
2. $f^{-1}(\beta(m)) = \bigcup_{i \in \omega} \alpha(H(m, i)) \cap \mathrm{dom}(f)$.

In the following if a partial computable function f is everywhere defined we say f is a *computable function*. It is easy to see that computable functions are effectively continuous and map a computable element to a computable element (c.g. [27]).

Remark 1. It is worth noting that the weakly effective ω–continuous domains and computable Polish spaces with induced topologies are proper subclasses of effectively enumerable T_0–spaces [17]. Therefore for uniformity we consider all those spaces in the settings of the effectively enumerable T_0–spaces.

3 Main Results

In this section we first computably embed a computable Polish space $\mathbb{P}$ into a lifted domain $\mathbb{D}^{\mathbb{P}}$ that is a weakly effective ω–continuous domain such that the Scott topology on $\mathbb{D}^{\mathbb{P}}$ agrees with the standard topology on $\mathbb{P}$ and the proposed embedding is a homeomorphism between the set of maximal elements and the original space. We prove that lifted domain presentations are computably and topologically minimal. We establish a criterion of computable minimality of any effective domain presentations. Then we show that while all topologically minimal presentations of computable Polish spaces are not stable there exist principal translators in the case of the lifted domain presentation of the reals.

3.1 Effective (Continuous) Domain Presentations for CPS

Definition 5. *Let $\mathbb{P}$ be a computable Polish space. A triple $(\mathbb{P}, \mathbb{D}, \varphi)$ is called an effective (continuous) domain presentation if*

1. *$\mathbb{D} = (D; \mathbf{B}, \beta, \leq, \perp)$ is a weakly effective ω–continuous domain;*
2. *The function $\varphi : \mathbb{P} \to \mathbb{D}$ is a computable (continuous) homeomorphic embedding such that $\mathrm{im}(\varphi) = \bigcap_{n\in\omega} O_n$ for some computable sequence of effectively open sets $\{O_n\}_{n\in\omega}$.*

The definition has been motivated by observations and examples in [29, 1, 18]. Thus in [18] we proposed a computable homeomorphic embedding $\varphi : C[0,1] \to \mathbb{D}$, where $\mathbb{D}$ is a weakly effective ω–continuous domain consisting of all continuous functions from the compact $[0,1]$ to $\mathcal{I}_{\mathbb{R}}$. Remarkably it turns out that $\mathrm{dom}(\varphi) \subset max(D)$ and $(C[0,1], \mathbb{D}, \varphi)$ is an effective domain presentation.

We recall from [13] the construction and properties of lifted domains for computable Polish spaces. Let $\mathbb{P} = (P, d, \mathcal{B}) \in CPS$. Then the lifted domain $(\mathbb{D}^{\mathbb{P}}, \psi)$ for $\mathbb{P} \in CPS$ is defined as follows:
$D^{\mathbb{P}} \leftrightharpoons P \times \mathbb{R}^+ = \{(a, r) \mid a \in P \text{ and } r \in \mathbb{R}^+\}$; $(b, q) \leq (a, r) \leftrightharpoons d(a, b) + r \leq q$;
$\mathbf{B} \leftrightharpoons \{(a, q) \mid a \in \mathcal{B} \text{ and } q \in \mathbb{Q}^{>0}\}$ and the numbering $\beta : \omega \to \mathbf{B}$ is induced by $\alpha : \omega \to \mathcal{B}$ and the standard numbering of $\mathbb{Q}^{>0}$. It is easy to see that the way-below relation $\ll$ has the property $(b, q) \ll (a, r) \leftrightarrow d(a, b) + r < q$. and the sub-basis of the Scott topology $\tau_{D^{\mathbb{P}}}$ is the set of open sets $\mathcal{U}_{n,q} = \{(b, r) \mid (b, r) \gg (\alpha(n), q),$ where $\alpha(n) \in \mathcal{B}$ and $q \in \mathbb{Q}^{>0}\}$. The function ψ is defined as follows $\psi(a) = (a, 0)$.

Remark 2. On the real numbers $\mathbb{R}$ with the standard metric there is a one-to-one correspondence between the pair (a, r) and the closed ball $\overline{B}(a, r)$ and the relation $\leq$ coincides with the inverse non-strict inclusion, i.e., $(b, q) \leq (a, r) \leftrightarrow \overline{B}(a, r) \supseteq \overline{B}(a, r)$. Unfortunately, in general on a computable Polish space there is no such intuitive natural correspondence.

Proposition 2. *The lifted domain $(\mathbb{D}^{\mathbb{P}}, \psi)$ for $\mathbb{P} \in CPS$ has the following properties:*

1. *$\mathbb{D}^{\mathbb{P}} = (D^{\mathbb{P}}; \mathbf{B}, \beta, \leq, \perp)$ is a weakly effective ω–continuous domain;*
2. *$\psi : \mathbb{P} \to \mathbb{D}^{\mathbb{P}}$ is a computable canonical homeomorphic embedding;*
3. *$\mathrm{im}(\psi)$ is dense in $D^{\mathbb{P}}$ and coincides with the set of maximal elements;*
4. *$\mathrm{im}(\psi)$ is an effective intersection of effectively open sets;*
5. *$\mathbf{B} \cap \mathrm{im}(\psi) = \emptyset$.*

Proof. The claims follow from [13, 14].

Corollary 1. *Let $(\mathbb{D}^{\mathbb{P}}, \psi)$ be the lifted domain for $\mathbb{P} \in CPS$. Then $(\mathbb{P}, \mathbb{D}^{\mathbb{P}}, \psi)$ is an effective domain presentation.*

Further on we call $(\mathbb{P}, \mathbb{D}^{\mathbb{P}}, \psi)$ as a lifted domain presentation.

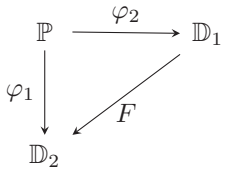
Proposition 3. *The interval domain $\mathcal{I}_{\mathbb{R}}$ is computationally isomorphic to the lifted domain presentation $(\mathbb{R}, \mathbb{D}^{\mathbb{R}}, \psi)$.*

Proof. The claim is straightforward from Remark 2.

3.2 Computable and Topological Minimality

In this section we assume that $\mathbb{P} = (P, d, \mathcal{B}) \in CPS$, $(\mathbb{D}^{\mathbb{P}}, \psi)$ is the corresponding lifted domain for $\mathbb{P}$. For the basic elements of a weakly effective ω–continuous domain $\mathbb{D} = (D; \mathbf{B}, \beta, \leq, \perp)$ we use the following notation $\mathbf{B} = \{\beta_1, \ldots, \beta_n, \ldots\}$.

Definition 6. *Let $(\mathbb{P}, \mathbb{D}_1, \varphi_1)$ and $(\mathbb{P}, \mathbb{D}_2, \varphi_2)$ be effective (continuous) domain presentations. A function $F : \mathbb{D}_1 \to \mathbb{D}_2$ is called a computable (continuous) translator if the following diagram is commutative:*



Definition 7. *An effective (continuous) domain presentation $(\mathbb{P}, \mathbb{D}, \psi)$ is called computably (topologically) minimal if for any effective (continuous) domain presentation $(\mathbb{P}, \widetilde{\mathbb{D}}, \widetilde{\psi})$ there exists a computable (continuous) translator $G : \mathbb{D} \to \widetilde{\mathbb{D}}$.*

Theorem 1. *For any computable Polish space $\mathbb{P}$ the lifted domain presentation $(\mathbb{P}, \mathbb{D}^{\mathbb{P}}, \psi)$ is computably minimal.*

The proof is based on the following propositions.

Lemma 1. *Let $\mathbb{D} = (D; \mathbf{B}, \beta, \leq, \perp)$ be a weakly effective ω–continuous domain and $\mathbb{X}$ be an effectively enumerable topological space. If a function $f : \mathbb{X} \to \mathbb{D}$ is computable then the following accessions hold.*

1. *Let $A_\beta = f^{-1}(\mathcal{U}_\beta)$, where $\mathcal{U}_\beta = \{d \in D \mid d \gg \beta\}$. Then $\{A_\beta\}_{\beta \in \mathbf{B}}$ is a computable sequence of effectively open subsets of X such that, for all $\beta, \gamma \in \mathbf{B}$, $A_\beta = \bigcup_{\beta' \gg \beta} A_{\beta'}$, $A_\beta \cap A_\gamma = \bigcup_{\beta' \gg \beta \wedge \beta' \gg \gamma} A_{\beta'}$ and if $\beta \leq \gamma$ then $A_\beta \supseteq A_\gamma$.*
2. *$f(x) = \bigsqcup\{\beta \in \mathbf{B} \mid x \in A_\beta\}$.*

Proof. Let us show the first accession. Computability of the sequence $\{A_\beta\}_{\beta \in \mathbf{B}}$ follows from computability of f. The relation $A_\beta \supseteq \bigcup_{\beta' \gg \beta} A_{\beta'}$ is straightforward. Assume now that $x \in A_\beta$. By definition, $f(x) \in \mathcal{U}_\beta$, i.e., $f(x) \gg \beta$. By the interpolation property there exists $\beta' \in \mathbf{B}$ such that $f(x) \gg \beta' \gg \beta$. So $x \in A_{\beta'}$, $x \in \bigcup_{\beta' \gg \beta} A_{\beta'}$. The relation $A_\beta \cap A_\gamma \supseteq \bigcup_{\beta' \gg \beta \wedge \beta' \gg \gamma} A_{\beta'}$ is straightforward. In order to show $A_\beta \cap A_\gamma \subseteq \bigcup_{\beta' \gg \beta \wedge \beta' \gg \gamma} A_{\beta'}$ we assume $x \in A_\beta \cap A_\gamma$. By definition, $f(x) \gg \beta$ and $f(x) \gg \gamma$. By the interpolation property and computability of f there exists $\beta' \in \mathbf{B}$ such that $f(x) \gg \beta'$ and $\beta' \gg \beta \wedge \beta' \gg \gamma$. So $x \in A_{\beta'}$.

The second assertion follows from the following observation. On the one hand, if $x \in A_\beta$ then $f(x) \gg \beta$, so $f(x) \geq \bigsqcup\{\beta \in \mathbf{B} \mid x \in A_\beta\}$. On the another hand, if $f(x) \gg \beta'$ then $x \in A_{\beta'}$, so $\beta' \leq \bigsqcup\{\beta \in \mathbf{B} \mid x \in A_\beta\}$. Since $f(x) = \bigsqcup\{\beta' \mid \beta' \ll f(x)\}$ we have $f(x) \leq \bigsqcup\{\beta \in \mathbf{B} \mid x \in A_\beta\}$.

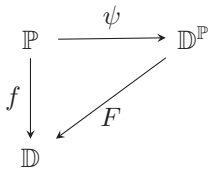
Lemma 2. *Let $\mathbb{D} = (D; \mathbf{B}, \beta, \leq, \perp)$ be a weakly effective ω–continuous domain and $\mathbb{X}$ be an effectively enumerable topological space and $\{A_\beta\}_{\beta \in \mathbf{B}}$ be a computable sequence of effectively open sets open subsets of X such that*

1. *If $\beta_1 \leq \beta_2$ then $A_{\beta_1} \supseteq A_{\beta_2}$.*
2. *For all $\beta, \gamma \in \mathbf{B}$, $A_\beta = \bigcup_{\beta' \gg \beta} A_{\beta'}$ and $A_\beta \cap A_\gamma = \bigcup_{\beta' \gg \beta \wedge \beta' \gg \gamma} A_{\beta'}$. Then the function $F : \mathbb{X} \to \mathbb{D}$ defined as follows $F(x) = \bigsqcup\{\beta \in \mathbf{B} \mid x \in A_\beta\}$ is computable. Moreover, $A_\beta = F^{-1}(\mathcal{U}_\beta)$.*

Proof. It is sufficient to show that $x \in F^{-1}(\mathcal{U}_\beta) \leftrightarrow F(x) \gg \beta$. If $x \in A_\beta$ then there exists $\beta' \gg \beta$ such that $x \in A_{\beta'}$. and $F(x) \gg \beta' \gg \beta$, i.e., $F(x) \gg \beta$. If $F(x) \gg \beta$ then there exists $\beta' \gg \beta$ such that $x \in A_{\beta'}$ and $\beta' \geq \beta$, so $x \in A_\beta$.

Theorem 2. *Let $\mathbb{D} = (D; \mathbf{B}, \beta, \leq, \perp)$ be a weakly effective ω–continuous domain, $\mathbb{P}$ be a computable Polish space and $(\mathbb{P}, \mathbb{D}^{\mathbb{P}}, \psi)$ be its lifted domain. Then for any computable function $f : \mathbb{P} \to \mathbb{D}$ one can effectively construct a computable extension $F : \mathbb{D}^{\mathbb{P}} \to \mathbb{D}$ such that*

1. *$F\big(\psi(x)\big) = f(x)$.*
2. *The following diagram is commutative:*



Proof. The claim follows from Lemma 1–2. Indeed, let $f : \mathbb{P} \to D$ be a computable function. By Lemma 1 we construct a computable sequence $\{A_\alpha\}_{\alpha \in \mathbf{B}_{\mathbb{D}}}$ of effectively open subsets of $\mathbb{P}$ that satisfies all properties from Lemma 1. Then, applying thee canonical embedding ψ we get a computable sequence $\{\mathcal{O}_\alpha\}_{\alpha \in \mathbf{B}_{\mathbb{D}}}$ of effectively open subsets of $\mathbb{D}^{\mathbb{P}}$ such that

1. For all $\beta, \gamma \in \mathbf{B}_{\mathbb{D}}$, $\mathcal{O}_\beta \cap \mathcal{O}_\gamma = \bigcup_{\delta \gg \beta,\, \delta \gg \gamma} \mathcal{O}_\delta$.
2. If $\beta_1 \leq \beta_2$ then $\mathcal{O}_{\beta_1} \supseteq \mathcal{O}_{\beta_2}$.
3. $\bigcup_{(a,r) \in \mathcal{O}_\alpha} B(a, r) = A_\alpha$.

Using $\{\mathcal{O}_\alpha\}_{\alpha \in \mathbf{B}_{\mathbb{D}}}$ and Lemma 2, we can effectively construct the computable function $F : \mathbb{D}^{\mathbb{P}} \to \mathbb{D}$. From the property (3) of the computable sequence $\{\mathcal{O}_\alpha\}_{\alpha \in \mathbf{B}_{\mathbb{D}}}$ it follows that for all $x \in \mathbb{P}$ and $\alpha \in \mathbf{B}_{\mathbb{D}}$ $f(x) \gg \alpha \leftrightarrow F(\psi(x)) \gg \alpha$. Therefore $f(x) = F(\psi(x))$, so the required diagram is commutative.

Remark 3. It is worth noting that the statement of the previous theorem holds not only for any lifted domain presentation but also for any effective domain presentation $\varphi : \mathbb{P} \to \mathbb{D}_0$ with φ^{-1} is pcf.

Corollary 2. *For any computable Polish space $\mathbb{P}$ the lifted domain presentation $(\mathbb{P}, \mathbb{D}^{\mathbb{P}}, \psi)$ is topologically minimal.*

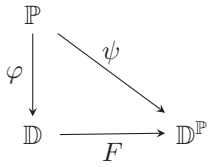
Proof. The claim follows from the relativization of Theorem 1 to an oracle making a lifted domain presentation computably minimal.

3.3 Criterion of Computable Minimality

Theorem 3. *For an effective domain presentation $(\mathbb{P}, \mathbb{D}, \varphi)$ the following assertions are equivalent.*

1. *$(\mathbb{P}, \mathbb{D}, \varphi)$ is computably minimal.*
2. *$g = \varphi^{-1}$ is pcf.*

Proof. 1) $\rightarrow$ 2). By definition we have the commutative diagram



where $F : \mathbb{D} \rightarrow \mathbb{D}^{\mathbb{P}}$ is a computable translator. Since the function $h \equiv \psi^{-1}$ is pcf, we can put g be equal to the restriction of the composition $h \circ F$ to the set $\mathrm{im}(\varphi)$. From [14] it follows that g is also pcf and $g = \varphi^{-1}$.
2) $\rightarrow$ 1). The claim follows from Theorem 2 and Remark 3.

In order to illustrate how this criterion works we consider the function domain $\mathcal{I}_f[0,1] = \{f \mid f : [0,1] \rightarrow \mathcal{I}_{\mathbb{R}} \text{ is continuous}\}$ and the considered in [18] natural homeomorphic embedding $\varphi : C[0,1] \rightarrow \mathcal{I}_f[0,1]$. From [18] it follows that φ has the inverse $g = \varphi^{-1}$ that is pcf. Therefore $(C[0,1], \mathcal{I}_f[0,1], \varphi)$ is computably minimal. Our considerations above revel the following properties of the interval domain for real numbers that widely used in domain theory and interval computations.

Theorem 4. *Let $(\mathbb{P}_1, \mathbb{D}^{\mathbb{P}_1}, \psi_1)$ be a lifted domain presentation and $(\mathbb{R}, \mathcal{I}_{\mathbb{R}}, \psi_2)$ be the interval domain for the reals. For any pcf $f : \mathbb{P}_1 \rightarrow \mathbb{R}$ there exists a total computable function $F : \mathbb{D}^{\mathbb{P}_1} \rightarrow \mathcal{I}_{\mathbb{R}}$ such that*

1. *$f(x) = y \leftrightarrow F(\psi_1(x)) = \psi_2(y) \wedge (x \in \mathrm{dom}(f))$;*
2. *if $x \notin \mathrm{dom}(f)$ then $F(\psi_1(x)) \notin max(\mathcal{I}_{\mathbb{R}})$.*

Proof. Let $f : \mathbb{P}_1 \rightarrow \mathbb{R}$ be pcf. In [12] we have shown that for the class of real-valued functions from computable metric spaces the notion of pcf coincides with majorant-computability. That means that we can effectively construct effectively open sets $U(x,y)$ and $V(x,y)$ such that $V(x,\cdot) < U(x,\cdot)$ and

$$f(x) = y \leftrightarrow \forall z_1 \forall z_2\, (V(x,z_1) < y < U(x,z_2)) \wedge$$
$$\{z \mid V(a,z)\} \cup \{z \mid U(a,z)\} = \mathbb{R} \setminus \{y\}.$$

Now we define $H : \mathbb{P}_1 \rightarrow \mathcal{I}_{\mathbb{R}}$ as follows: Put

$$H(x) = \begin{cases} [\sup\{y \mid V(x,y), \inf\{z \mid U(x,z)\}] & \text{if } V(x,\cdot),\, U(x,\cdot) \neq \emptyset \\ \bot, & \text{otherwise.} \end{cases}$$

It is easy to see that H is a computable function and, for all $x \in \mathbb{P}$, $H([x]) = f(x)$. Then the existence of F follows from Theorem 2.

3.4 Principal Translators

In this section we introduce the notion of a principal computable (continuous) translator and prove that in the case of the real numbers we can effectively construct a principal computable translator from the lifted domain presentation to any other domain presentation.

Definition 8. *Let $\mathbb{P}$ be a computable Polish space and $(\mathbb{P}, \mathbb{D}_1, \varphi_1)$ and $(\mathbb{P}, \mathbb{D}_2, \varphi_2)$ be its effective (continuous) domain presentations. A computable (continuous) translator $G : \mathbb{D}_1 \to \mathbb{D}_2$ is called principal if for any computable (continuous) translator $F : \mathbb{D}_1 \to \mathbb{D}_2$ we have $F \leq G$.*

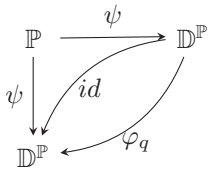
Definition 9. *For a computable Polish space $\mathbb{P}$, a continuous (computable) domain presentation $(\mathbb{P}, \mathbb{D}, \varphi)$ is called (computably) stable if for any continuous (computable) domain presentation $(\mathbb{P}, \widetilde{\mathbb{D}}, \widetilde{\psi})$ there exists a unique continuous (computable) translator $G : \mathbb{D} \to \widetilde{\mathbb{D}}$.*

Proposition 4. *For any computable Polish space the lifted domain presentation is neither computably stable nor stable.*

Proof. It is sufficient to show that there are infinitely many continuous even computable translators for $\widetilde{\mathbb{D}} \equiv \mathbb{D}^{\mathbb{P}}$. Put

$$G \equiv \mathrm{id} \text{ and } \varphi_q\big((a, r)\big) = (a, q * r), \text{ where } q \in \mathbb{Q}^+ \text{ and } q > 1.$$

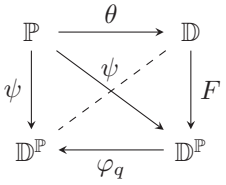
We have:



All of these translators are computable and different from each other.

Proposition 5. *If a continuous domain presentation $(\mathbb{P}, \mathbb{D}, \theta)$ is topologically (computably) minimal then it is not (computably) stable.*

Proof. Let $\mathbb{D} = (D; \mathbf{B}, \beta, \leq, \perp)$. It is enough to observe topological part, the rest is just an analog. Assume $F : \mathbb{D} \to \mathbb{D}^{\mathbb{P}}$ is a continuous translator. We have the following commutative diagram:



Since $F(\mathbf{B}) \not\subseteq max(D^{\mathbb{P}})$, there exists $a \in \mathbf{B}$ such that $F(a) \notin max(\mathbb{D}^{\mathbb{P}})$. Then $\varphi_q(F(a)) \neq F(a)$, where φ is defined in Proposition 4. We have $\varphi_q \circ F : \mathbb{D} \to \mathbb{D}^{\mathbb{P}}$ is a continuous translator and $\varphi_q \circ F \not\equiv F$.

Theorem 5. *Let $(\mathbb{P}, \mathbb{D}^{\mathbb{P}}, \psi)$ be the lifted domain presentation and $(\mathbb{P}, \widetilde{\mathbb{D}}, \widetilde{\psi})$ be a continuous domain presentation such that $\widetilde{\mathbb{D}}$ is a complete upper semilattice (cusl). Then there exists a principal continuous translator $G : \mathbb{D}^{\mathbb{P}} \to \widetilde{\mathbb{D}}$.*

Proof. Let us define

$$G\big((a,r)\big) = \bigsqcup\{g\big((a, r + \frac{1}{n})\big)\}_{n\in\omega}, \text{ where } g\big((a,r)\big) = \inf\{\widetilde{\psi}(x) \mid (a,r) \leq \psi(x)\}.$$

From the definition of ψ it is easy to see that $g\big((a,r)\big) = \inf\{\widetilde{\psi}(x) \mid (a,r) \leq (x,0)\} = \inf\{\widetilde{\psi}(x) \mid x \in \overline{B}(a,r)\}$. In order to show that G is a required we prove that G is total and monotone, preserves limits and makes the diagram commutative.

Totality. It is worth noting that for any $Y \subseteq \widetilde{D}$ there exists $\inf(Y)$. Indeed, since $\widetilde{\mathbb{D}}$ is cusl the set $\{z \mid z \leq Y\}$ is directed. Therefore $\inf(Y) = \bigsqcup\{z \mid z \leq Y\}$ for $Y \neq \emptyset$ and $\inf(\emptyset) = \bot$ by the definition of dcpo.

Monotonicity. By definition it is clear that g is monotone Assume $(b,R) \leq (a,r)$, i.e., $d(a,b) + r \leq R$. By definition, $G\big((a,r)\big) = \bigsqcup \inf\{\widetilde{\psi}(x) \mid x \in \overline{B}(a, r + \frac{1}{n})\}_{n\in\omega}$, $G\big((b,R)\big) = \bigsqcup \inf\{\widetilde{\psi}(x) \mid x \in \overline{B}(b, R + \frac{1}{n})\}_{n\in\omega}$ and, by assumption, $(b, R + \frac{1}{n}) \leq (a, r + \frac{1}{n})$. Therefore $G\big((b,R)\big) \leq G\big((a,r)\big)$.

Limit preservation. Since $\widetilde{\mathbb{D}}$ is a weakly effective ω–continuous domain to prove that G preserves limits it is sufficient to consider countable directed sets. We show first that for any directed sets $\mathcal{A},\ \mathcal{B} \subseteq D$, if $\bigsqcup \mathcal{A} = \bigsqcup \mathcal{B} = (a,r)$ and $\mathcal{A} \ll (a,r)$, $\mathcal{B} \ll (a,r)$ then $\bigsqcup\{g\big((a,r)\big) \mid (a,r) \in \mathcal{A}\} = \bigsqcup\{g\big((b,R)\big) \mid (b,R) \in \mathcal{B}\}$ that looks as the low semi-continuity condition. Let us pick a basic elements $\beta \in \widetilde{\mathbf{B}}$ such that $\beta \ll \bigsqcup\{g\big((a,r)\big) \mid (a,r) \in \mathcal{A}\}$. By the definition of the way-below relation, there exists $(a,r) \in \mathcal{A}$ such that $\beta \ll g\big((a,r)\big)$ so for all $x \in \overline{B}(a,r)$ we have $\beta \ll \widetilde{\psi}(x)$. Since $\bigsqcup \mathcal{A} = \bigsqcup \mathcal{B}$ there exists $(b,R) \geq (a,r)$ so for all $x \in \overline{B}(b,R)$ we have $\beta \ll \widetilde{\psi}(x)$. This means $\beta \ll \bigsqcup\{g\big((b,R)\big) \mid (b,R) \in \mathcal{B}\}$. Since β is arbitrary chosen $\bigsqcup\{g\big((a,r)\big) \mid (a,r) \in \mathcal{A}\} \geq \bigsqcup\{g\big((b,R)\big) \mid (b,R) \in \mathcal{B}\}$. By symmetry, $\bigsqcup\{g\big((a,r)\big) \mid (a,r) \in \mathcal{A}\} = \bigsqcup\{g\big((b,R)\big) \mid (b,R) \in \mathcal{B}\}$.

Now assume that, for a countable directed set $\mathcal{A} \subseteq D$, $\bigsqcup \mathcal{A} = (a,r)$. It is well-known that we can extract some monotone sequence $\{(a_n, r_n)\}_{n\in\omega}$ of elements of $\mathcal{A}$ such that $\bigsqcup\{(a_n,r_n)\}_{n\in\omega} = (a,r)$. Therefore it is sufficient to prove that $\bigsqcup\{G\big((a_n,r_n)\big)\}_{n\in\omega} = G(a,r)$. By definition $G\big((a_n,r_n)\big) = \bigsqcup\{(a_n, r_n + \frac{1}{k}\}_{k\in\omega}$. It is easy to see that $\bigsqcup\{(a_n, r_n + \frac{1}{k})\}_{n,k\in\omega} = (a,r)$ and $(a_n, r_n + \frac{1}{k}) \ll (a,r)$ for all $n,\ k \in \omega$. By the property of g which we proved above $\bigsqcup\{g\big((a_n, r_n + \frac{1}{k})\big)\}_{n,k\in\omega} = \bigsqcup\{g\big(a, r + \frac{1}{m}\big)\}_{m\in\omega}$ so $\bigsqcup G\big((a_n,r_n)\big) = G\big((a,r)\big)$. Therefore G is a continuous function.

Commutativity of the diagram. We show that $\widetilde{\psi}\big(x\big) = G\big(\psi(x)\big)$, i.e, $\widetilde{\psi}\big(x\big) = G\big((x,0)\big)$ since $\psi(x) = (x,0)$. By definition, $G\big((x,0)\big) = \bigsqcup\{g\big((x, \frac{1}{n})\big)\}$ and, for

all $n \in \omega$, $g(x, \frac{1}{n}) \leq \widetilde{\psi}(x)$. Therefore $\widetilde{\psi}(x) \geq G\big((x,0)\big)$. Since $\widetilde{\psi}$ is continuous in x for all $\widetilde{\beta} \in \widetilde{\mathbf{B}}$ such that $\widetilde{\beta} \ll \widetilde{\psi}(x)$ there exist $\sigma > 0$ such that for all $y \in P$ if $d(y,x) < \sigma$ then $\widetilde{b} \ll \widetilde{\psi}(y)$. It is worth noting that if $\frac{1}{n} < \sigma$ then $g(x, \frac{1}{n}) \leq \widetilde{\beta}$. So we have $G\big((x,0)\big) \geq \widetilde{\beta}$. By continuity, $\widetilde{\psi}(x) = \bigsqcup\{\widetilde{\beta} \mid \widetilde{\beta} \ll \widetilde{\psi}(x)\}$ so $\widetilde{\psi}(x) \leq G\big((x,0)\big)$.

Maximality. Let us show that for any continuous translator $F : \mathbb{D}^{\mathbb{R}} \to \widetilde{\mathbb{D}}$ we have $F \leq G$. First we observe that if $x \in max(\mathbb{D}^{\mathbb{P}})$ then $F(x) = G(x)$. By monotonicity of F, $F\big((a,r)\big) \leq \inf\{\widetilde{\psi}(x) \mid (a,r) \leq \psi(x)\} = g\big((a,r)\big)$. Similarly, $F\big((a, r+\frac{1}{n})\big) \leq g\big((a, r+\frac{1}{n})\big)$. Since $G\big((a,r)\big) = \bigsqcup\{g\big((a, r+\frac{1}{n})\big)\}_{n\in\omega}$ and $F\big((a,r)\big) = \bigsqcup\{F\big((a, r+\frac{1}{n})\big)\}_{n\in\omega}$ we have $F\big((a,r)\big) \leq G\big((a,r)\big)$. As a corollary G is a required function.

Theorem 6. *Let $(\mathbb{R}, \mathbb{D}^{\mathbb{R}}, \psi)$ be the lifted domain presentation and $(\mathbb{R}, \widetilde{\mathbb{D}}, \widetilde{\psi})$ be an effective domain presentation such that $\widetilde{\mathbb{D}}$ is a cusl. Then there exists a principal computable translator $G : \mathbb{D}^{\mathbb{R}} \to \widetilde{\mathbb{D}}$.*

Proof. Let us show that in the case of $\mathbb{P} = \mathbb{R}$, the continuous function G from the previous proof is computable under the assumption that $\widetilde{\psi}$ is computable.

Computability. In order to show that G is computable it is sufficient to show that $F(\beta_k) \gg \widetilde{\beta}_m$ is computably enumerable. First we assume that $\beta_k = (a_k, r_k)$ and observe that the relation $g\big((a_k, r_k)\big) \gg \widetilde{\beta}_m$ is computable enumerable. It follows from the following formula and the uniformity principle [16].

$$g\big((a_k,r_k)\big) \gg \widetilde{\beta}_m \leftrightarrow (\forall x \in \overline{B}(a_k,r_k))\, \widetilde{\psi}(x) \gg \widetilde{\beta}_m \leftrightarrow (\forall x \in \overline{B}(a_k,r_k))\, x \in \widetilde{\psi}^{-1}(\mathcal{U}_{\widetilde{\beta}_m}).$$

Since, by definition, $G\big((a_k,r_k)\big) = \bigsqcup\{g\big((a_k, r_k+\frac{1}{n})\big)\}_{n\in\omega}$ we have

$$G\big((a_k,r_k)\big) \gg \widetilde{\beta}_m \leftrightarrow (\exists n \in \omega)\, g\big((a_k, r_k + \frac{1}{n})\big) \gg \widetilde{\beta}_m.$$

Therefore the required relation is computably enumerable and G is computable.

4 Conclusion and Future Work

In this paper we characterised computably minimal presentations of computable Polish spaces. We showed that between any computably minimal presentations one can effectively construct a translator. This gives a technique to convert one computably minimal presentation to another. Therefore a user can chose any preferable computably minimal presentation and then if necessary convert to canonical one. As a corollary we obtained that the effective domain of continuous functions on a compact interval is convertible to the formal ball presentation and vise versa. For the lifted domain of real numbers we provided a principal computable translator. This highlighted a direction of how to approach a formalisation of higher type computations over the reals.

References

1. Edalat, A. and Heckmann, R. (1998) A Computational Model for Metric Spaces. *Theor. Comput. Sci.* **193** (1-2), 53–73.
2. Edalat, A. (1997) Domains for computation in mathematics, physics and exact real arithmetic. *Bulletin of Symbolic Logic* **3** (4), 401–452.
3. Ershov, Yu. L. Computable functionals of finite types, *Algebra and Logic* **11** (4), 1996 pages 367-437.
4. Blanck, J. (2013) Interval Domains and Computable Sequences: A Case Study of Domain Reductions. *The Computer Journal* **56** (1), 45-52.
5. Blanck, J. (1997) Domain Representability of Metric Spaces. *Ann. Pure Appl. Logic* **83** 3, 225–247.
6. Brattka, V. (2001) Computable Versions of Baire's Category Theorem. In *MFCS'99, Lecture Notes in Computer Science* **2136**, 224-235. Springer.
7. Calvert, W., Fokina, E., Goncharov, S. S., Knight, J. F., Kudinov, O. V., Morozov, A. S. and Puzarenko, V. (2007) Index sets for classes of high rank structures. *J. Symb. Log.* **72** (4), 1418-1432.
8. Calvert, W., Harizanov, V. S., Knight, J. F., Miller, S. (2006) Index sets of computable structures. *J. Algebra and Logic* **45** (5), 306–325.
9. Cenzer, D.A., Remmel, J.B. (1999) Index Sets in Computable Analysis. *T*heor. Comput. Sci. **219** (1-2), 111-150.
10. Gierz, G., Heinrich Hofmann, K., Keime, lK., Lawson, J. D. and Mislove, M. W. (2003) *Continuous Lattices and Domain*. Encyclopedia of Mathemtics and its Applications 93, Cambridge University Press.
11. Grubba, T., Weihrauch, K. (2009) Elementary Computable Topology. *J. UCS.* **15** (6), 1381–1422.
12. Weak Reduction Principle and Computable Metric Spaces (2018) In *Proc. CiE'18, Lecture Notes in Computer Science*, **10936**. Springer.
13. Korovina, M. and Kudinov, O. (2018) Highlights of the Rice-Shapiro Theorem in Computable Topology. In *Postproc. PSI'17, Lecture Notes in Computer Science* **10742**, 241–255. Springer.
14. Korovina, M. and Kudinov, O. (2017) Outline of Partial Computability in Computable Topology. (invited paper) In *Proc. CiE'17, Lecture Notes in Computer Science* **10307**, 64–76. Springer.
15. Korovina, M. and Kudinov, O. (2015) Index sets as a measure of continuous constraints complexity. In *Proc. PSI'14, Lecture Notes in Computer Science* **8974**, 201–215. Springer.
16. Korovina, M. and Kudinov, O. (2009) The Uniformity Principle for Sigma-definability. *J. Log. Comput.* **19** (1), 159–174.
17. Korovina, M. and Kudinov, O. (2008) Towards Computability over Effectively Enumerable Topological Spaces. *Electr. Notes Theor. Comput. Sci.* **221**, 115–125.
18. Korovina, M. and Kudinov, O. (2001) Formalisation of Computability of Operators and Real-Valued Functionals via Domain Theory. *Lecture Notes in Computer Science* **2064**, 146–168.
19. Mal'Cev, A. (1961) Constructive algebras. Uspehi Math Nauk, **16** (3), 3–60.
20. Moschovakis, Y. N. (1964) Recursive metric spaces. *Fund. Math.* **55**, 215–238.
21. Rabin, M. (1960) Computable algebra, general theory and theory of computable fields. *Transactions of the American Mathematical Society* **95**, 341360.
22. Rogers, H. (1967) *Theory of Recursive Functions and Effective Computability* McGraw-Hill, New York.

23. Scott, D. (1982) Lectures on a Mathematical Theory of Computation. *Theoretical Foundations of Programming Methodology*, 145-292.
24. Soare, R. I. (1987) *Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets*. Springer.
25. Spreen, D. (1995) On Some Decision Problems in Programming. *Inf. Comput.* **122** (1), 120-139.
26. Spreen, D. (1998) On Effective Topological Spaces. *J. Symb. Log.* **63** (1), 185–221.
27. Weihrauch, K. (2000) *Computable Analysis*. Springer Verlag.
28. Weihrauch, K. (1993) Computability on Computable Metric Spaces. *Theor. Comput. Sci.* **113** (1), 191–210.
29. Weihrauch, K. and Schreiber, U. (1981) Embedding metric spaces into cpos. *Theoret. Comput. Sci.* **16**, 5-24.

About Leaks of Confidential Data in the Process of Indexing Sites by Search Crawlers[1]

Sergey Kratov[0000-0001-9068-9267]

Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, Russia
kratov@sscc.ru

Abstract. The large number of sites for very different purposes (online stores, ticketing systems, hotel reservations, etc.) collect and store personal information of their users, as well as other confidential data, such as history and results of user interaction with these sites. Some of such data, not intended for open access, nevertheless falls into the search output and may be available to unauthorized persons when specific requests are made. This article describes the reasons for such incidents occurrence and the basic recommendations for technical specialists (developers and administrators) that will help prevent leaks.

Keywords: Data Leaks, Site Indexing, Search Crawlers, Robots.txt, Noindex, X-Crawlers-Tag, Htaccess.

1 The Possible Sources of Data Leaks

One of the factors determining the effectiveness of the search is the completeness of the index. So the search engines try to index as many pages as possible to select those that most closely match the users' requests. Therefore, in addition to going through the links on the pages, the search engines also resort to other methods that allow it to discover the appearance of new pages on the Internet. Very often sites generate individualized pages for each user. So that search engines cannot get to seemingly public pages from links on the main page of the site. Accordingly, it is logical for search engines to obtain page addresses for indexing from as many sources as possible. In particular, for example, users agree by default with possible analysis and collection of browsers anonymous data on page visits and other actions, when installing browsers, often developed by search engines (Yandex, Google). This is the legal way for search engines to collect most of the pages ever viewed by users. For example, Yandex.Browser collects anonymized statistical information, which includes, in addition, the addresses of the pages visited. This happens in all cases when the user clearly did not forbid doing this in the browser settings (the option "Send usage statistics to Yandex"). At the same time, the-

[1] The research was supported by the project 0315-2016-0006.

matic forums for developers described situations when, due to a technical error of Yandex employees, information about individual pages viewed in the browser came to the list of indexing for Yandex crawlers.

Another source of data for the search engines index replenishment can be the counters of analytical systems (the most common ones in Russia are Yandex.Metrica and Google Analytics) placed in the page source code. For example, as specified in the Yandex.Metrica user agreement [1] if site administrators do not forbid sending site pages to indexing, the addresses of the pages on which the counter is installed are passed to Yandex indexing (and it is possible subsequently to the search output). By default, such option is enabled. That is how a few years ago SMS from the mobile operator Megafon's site fells in the search output. There was the possibility of anonymous sending SMS on the operator's site to its clients. This did not require registration on the site. At the same time, the site developers, for the convenience of users, generated for each sending a page with a random address, which displayed the SMS text and the status of its delivery. These pages got into Yandex's search output and became available to any user of the search engine.

Only the couple of data leakage cases were listed above on the example of the one of the search engines services. Nevertheless, there is no reason to suppose that the rest of the search engines are fundamentally different. The issues of search engines legality and their crawlers «ethical» nature in the data collection process have already been repeatedly discussed in the other researchers' works [2-4].

2 The Informing Developers and Users about the Possibility of Leaks

Search crawlers cannot access and indexing information from pages that require authorization. At the same time, modern sites often require complex passwords, which are not always convenient for users to remember. For the convenience of users, developers often generate and send to users in emails (in plain text) links to pages with unique long addresses from a random character set that cannot be guessed or enumerated. So users can grant direct access to sites without entering a username and password. Users navigate through such link, their browser or analytics counter tells the search engine that an unknown page has appeared, the search crawler indexes it. In this case, the crawler has no information about whether the personal data is placed on the page, whether confidential information is contained in tables (for example, financial indicators) or the content of the page is publicly available. Useful recommendations for developers who are forced to generate and send pages with automatic login are available in the corresponding W3C manual [5].

Confidential data periodically fell into search indexes during the entire existence of search engines. The number of such leaks, increased in recent years, is associated with the growing popularity of the Internet and, accordingly, the number of users of the network. More and more people are entering the network, now they are not only IT specialists, but also users far from information technologies. Most users believe that a document accessible via a unique link is securely protected and will never get into the

index. The main changes to minimize the number of similar incidents in the future should be done by site developers to ensure their quality work. Search engines, in turn, should also fully cover their indexing mechanisms for both developers and site users. In particular, to inform developers that any page that is available to users without authorization can sooner or later get into the index and search output. The last such large-scale data leak in the Russian-speaking Internet segment occurred in early July 2018. The search engine Yandex indexed and included in its search output a large array of documents from the Google Docs service. The documents were publicly available and, accordingly, were available for indexing, but at the same time, many of them contained confidential information not intended for unauthorized persons. Moreover, some of the documents were not only available for viewing, but also for editing to any user who passed them by link from the index. Formally, both Yandex and Google in this situation acted within the law. The documents were excluded from the search output in identifying the problem. The problem arose primarily because of the lack of users' awareness about the specifics of the access differentiation to their documents in the service. But this fact does not cancel the presence of the problem itself.

3 The Prohibition of Confidential Data Indexing. Directives for Search Crawlers

The most effective way to deny access to confidential information is to use authorization to access it. However, in those cases when it is impossible or impractical for not to complicate the work of users with the site, developers can use other methods that will prevent search crawlers from indexing the contents of the pages and in many ways will reduce the probability of their getting into the search output. For example, it is possible to use the robots.txt file or corresponding tags in the HTML markup and page headers [6-8].

The standard of exclusions for crawlers Robots.txt was adopted by the W3C in 1994 and has since been supported by most search engines. The standard is the text file describing the limitations of the search crawlers' access to the web server's content. This file should be uploaded to the site's root directory. The file consists of separate records, the Disallow field is used to prevent indexing. With this field, developers can deny access to individual directories / pages for all or individual search crawlers. The example of the appropriate entry completely closing access to the site for search crawlers:

```
User-agent: *
Disallow: /
```

In the User-agent field, individual search crawlers can be enumerated, for example, Yandex or Googlebot. In the Disallow field, you can specify both the name of the individual file and the directory as a whole. More details about the syntax of robots.txt and practical recommendations for its use can be read in the relevant standard [9] and earlier studies on this topic [10, 11]. It is also recommended after creating or editing the file to check its syntax correctness by using search engines special services [12, 13].

In cases where site administrators do not have access to the site root directory to host the robots.txt file, or site administrators do not want to advertise the individual directories/files addresses, the noindex tag can be used anywhere in the HTML code of the page:

```
<noindex> the text that does not need to be indexed </
noindex>
```

The noindex tag is not included in the official HTML specification so, if it is used in the code of the pages, they may fail to validate the html code correctness. In such cases the noindex tag can be used in the following format:

```
<! - noindex -> the text that does not need to be indexed
<! - / noindex ->
```

The noindex tag can also be used as the metatag, in which case its action will extend to the entire text of the page as a whole:

```
<meta name = "crawlers" content = "noindex" />
```

In addition, although Google does not index the contents of pages blocked in the robots.txt file, such pages URLs found on other pages on the Internet can still be indexed [14]. In this case, the use of noindex in the metatag form in the page header will further prohibit its indexing when a search crawler hits it.

The separate metatag can be requested to delete the previously indexed page copy from the search engine cache:

```
<meta name = "crawlers" content = "noarchive" />
```

The noindex metatag can be used only in the code of html-pages. If developers want to deny access to other types of documents, they can use the X-Crawlers-Tag metatag contained in the HTTP header. An example of the HTTP header that prohibits crawlers from indexing the page:

```
HTTP / 1.1 200 OK
...
X-Crawlers-Tag: noindex
...
```

If site works on the Apache web server, its administrators can insert the appropriate headers using directives in the .htaccess file. For example, the following directives prohibit search crawlers from indexing all pdf-files of the site:

```
<Files ~ "\ .pdf $">
Header set X-Crawlers-Tag "noindex, nofollow"
</ Files>
```

For more details about syntax and usage examples, see the appropriate Google manual [15].

4 Conclusion

Unfortunately, using the above methods does not guarantee that the site pages and individual files will not be indexed. These methods are recommendatory for search crawlers and the implementation or non-implementation of recommendations depends only on the particular crawler. The problem is that different search engines differently interpret the web servers' directives, their recommendations for improving the sites' indexing also often contradict each other. That is, the developers, having done everything according to the Google's instructions, can create a situation in which the Yandex will index a lot of documents that should not have been indexed, and vice versa. For example, Google does not handle noindex tags in the text of the pages, and Yandex - X-Crawlers-Tag in HTTP headers.

Therefore, administrators of already working sites need to conduct their basic audit for the leaking confidential data possibility to search engines:

- Carry out the entire tree of the site's links analysis - scan search output and other sources (Yandex.Metrics, Google Analytics, Yandex.Webmaster and Google Search Console). Identify the site's pages containing confidential data. Finding the reasons and determining how to hide these pages from indexing and from the publicly available part of the site.
- Analyze files, links to which are not present on the site pages - identify confidential files accessible via direct links, including those that are not yet in the search output. Search for the reasons for such files availability and determine how to hide them from public access.

In addition to the above actions to prohibit the confidential data's indexing, developers can also strongly encourage to take the following actions when creating new sites:

- Exclude any of the confidential data from sharing with authorization using.
- Identify search crawlers and block them from accessing any private information. And developers should not only use one of the methods recommended by any search engine, but duplicate, using all protection methods. Verify that the protection methods used are universal and workable for all search engines.
- Maximally inform users about all available privacy settings within each site.

References

1. Terms of Use of Yandex.Metrica service, https://yandex.ru/legal/metrica_termsofuse/, last accessed 2018/08/29.
2. Schellekens, M.H.M.: Are internet robots adequately regulated?. Computer Law and Security Review 29(6), 666-675 (2013). doi: 10.1016/j.clsr.2013.09.003
3. Sun, Y., Councill, I.G., Giles, C.L.: The ethicality of web crawlers. In: Proceedings - 2010 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2010, pp. 668-675. (2010). doi: 10.1109/WI-IAT.2010.316

4. Giles, C.L., Sun, Y., Councill, I.G.: Measuring the web crawler ethics. In: Proceedings of the 19th International Conference on World Wide Web, WWW '10, pp. 1101-1102. (2010). doi: 10.1145/1772690.1772824
5. Good Practices for Capability URLs, https://www.w3.org/TR/capability-urls/, last accessed 2018/08/29.
6. Martin-Galan, B., Hernandez-Perez, T., Rodriguez-Mateos, D. et al.: The use of robots.txt and sitemaps in the Spanish public administration. PROFESIONAL DE LA INFORMACION, vol. 18, iss. 6, 625-630 (2009). doi: 10.3145/epi.2009.nov.05
7. Kolay, S., D'Alberto, P., Dasdan, A., Bhattacharjee, A.: A larger scale study of robots.txt. In: Proceeding of the 17th International Conference on World Wide Web 2008, WWW'08, pp. 1171-1172. (2008). doi: 10.1145/1367497.1367711
8. Sun, Y., Zhuang, Z., Councill, I.G., Giles, C.L.: Determining bias to search engines from robots.txt. In: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, WI 2007, pp. 149-155. (2007). doi: 10.1109/WI.2007.45
9. A Standard for Robot Exclusion, http://www.robotstxt.org/orig.html, last accessed 2018/08/29.
10. Tong, W., Xie, X.: A research on a defending policy against the Webcrawler's attack. In: 2009 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication, ASID 2009 (2009). doi: 10.1109/ICASID.2009.5276948
11. Bates, M.E.: What makes information "public"?. Online (Wilton, Connecticut) 28(6), 64 (2009). doi: 10.1109/ICASID.2009.5276948
12. Robots.txt analysis, https://webmaster.yandex.ru/tools/robotstxt/, last accessed 2018/08/29.
13. robots.txt Tester, https://www.google.com/webmasters/tools/robots-testing-tool, last accessed 2018/08/29.
14. Blocking URLs with a robots.txt file, https://support.google.com/webmasters/answer/6062608, last accessed 2018/08/29.
15. Robots meta tag and X-Robots-Tag HTTP header specifications, https://developers.google.com/search/reference/robots_meta_tag, last accessed 2018/08/29.

Archival Information Systems: New Opportunities for Historians

Irina Krayneva, Sergey Troshkov

A.P. Ershov Institute of Informatics Systems,
Lavrentiev ave. 6, 630090 Novosibirsk, Russia
cora@iis.nsk.su, kamronis@xtech.ru

Abstract. This paper presents a brief summary of the twenty years of research carried out at the A. P. Ershov Institute of Informatics Systems SB RAS in the area of developing electronic archives for heterogeneous documents. The phenomenon of electronic archives emerged and has been developing as part of the Novosibirsk school of informatics, which has always been oriented towards the contracting of social services. In the 1970s, the first social service projects launched by the Institute were educational initiatives for school education, accordant with the well-known thesis of Andrey Ershov: "Programming is the second literacy". Over the years, the IIS SB RAS has completed a range of projects on digitizing historical and cultural heritage of the Siberian Branch of the Russian Academy of Sciences: the Academician A. P. Ershov Electronic Archive, SB RAS Photo Archive and the SB RAS Open Archive. This work has become especially relevant in view of the ongoing restructuring of the Russian academic science.

Keywords: A.P. Ershov · digital archives · digital historical factography · Drupal.

1 Introduction

The current information boom has brought up a number of challenges dealing with the problem of relevance of the information produced by a researcher; providing quality information to the scientific community has become a cornerstone task. James Nicolas Gray, an American researcher in computational systems and a Turing Award holder, suggested the concept of the fourth paradigm of research, a grid-technology based science that uses big data. Even though Gray and his followers stress the importance of systematization and free access to scientific archives (including experimental data and modeling results), the notion of a big virtual archive for humanities studies is no less relevant [1]. This highlights the relevance of free access to information since scientific workers are known to benefit from information and communication technologies (ICT) [2].

The demand for novelty and relevance of scientific research as a socio-cognitive institute stresses the need for a better and quicker access to archives, libraries, museums and other types of heritage content. Informatization as well as commercialization of state-run archives in Russia was propelled even further by the

emergence of the Internet in the 1990s. Evidently, there should be alternative options of free access to information as well. The IIS SB RAS research team was among the first to complete several projects on the creation, scientific interpretation, organization and development of the methodology of digitizing scientific heritage; our experience can be viewed as technology-intensive, scientifically based, successfully tested and implemented [3].

Interdisciplinary collaboration of specialists in human studies and IT at the dawn of the Internet relied predominantly on the concept of open scientific communications. In essence, this is a cluster of civil society supported by professionals. Museums, libraries, universities and research institutions getting access to the Internet boosted user experience and the emergence of Internet-oriented resources: published museum collections, online library systems and catalogs, archival tools for research and reference, and select collections. For instance, in the Novosibirsk Scientific Center, projects on the creation of electronic archives of different types of documents became possible as part of the project called "The Internet of Novosibirsk Scientific Center" in 1994–1998, funded by the Soros Foundation, Russian Foundation of Basic Research, and INTAS. As result, research institutions and other organizations of the NSC got free access to the Internet.

2 Tools

The emergence and distribution of special tools – information systems (IS) – facilitates the development and systematization of information in professional communities, including those engaged in humanities studies [4]. Electronic catalogs and knowledge bases became an integral part of scientific community processes by the end of the 20th century. Specialized information systems appeared aimed at presenting, storing and organizing historical sources and texts, i.e. history-oriented IS's. We understand information systems as a complex of technical, program, organizational and financial utilities as well as the personnel capable of working with this complex and complete the project [5]. The minimal staffing of such projects, based on the experience of the IIS SB RAS, is about 10 people, including programmers, historians, information specialists (operators) and translators.

Specialists from Perm State University whose interest is the application of the IT in humanities suggested a specification of history-oriented systems [6]. Of special interest are systems containing, in addition to historical information, a set of research tools (search, analytics, text recognition, etc.). We see two approaches to creating the IS's: history-oriented, when a system is based on an array of documents from a single source, and the system is modeled according to the structure of this source, and problem-oriented, when a model is built based on the structure of the studied field of knowledge. According to this classification, the systems created in the IIS SB RAS are history-oriented. The SB RAS Photo Archive integrates two sources on its platform: scans of photographic documents and archives of the *Nauka v Sibiri* (*Science in Siberia*) newspaper. There is an

organic and thematic connection between the two sources, because the newspaper staff reporters took many of the photos. In addition to documents, the A. P. Ershov's Electronic Archive and the SB RAS Open Archive contain photographs and research papers. We consider our IS's source-oriented for another reason, too: the archives contain images (scans) of original documents with transcriptions supplied as an additional feature allowing to read compromised and poorly legible scans. Finally, our IS's support remote workspaces for document description.

3 Analogies and Problems

Currently there are many resources created for the accumulation of historical and cultural heritage in a digital format. Millions of photographs from the LIFE photo archive, stretching from the 1750s to today, are now available for the first time through the joint work of LIFE and Google (2008). Digital collections of the Science History Institute (https://digital.sciencehistory.org/) includes 6,508 digitized items: artifacts, photographs, advertisements, letters, rare books. Library of Congress (https://www.loc.gov/) and digital collections of UNESCO (https://digital.archives.unesco.org/en/collection) are the most impressive ones. Though, they have no available catalogue to help determine the connections between documents.

One of the main problems faced by the creators of these projects was financing. In 2015, UNESCO launched a fundraising project to digitize the archives of the Organization belonging to its predecessors, including the League of Nations International Institute for Intellectual Cooperation. Two years later, thanks to the generous support of the Japanese government, UNESCO launched a major two-year initiative. In partnership with the digitization company Picturae BV, a laboratory was established at the site of UNESCO Headquarters in Paris in February 2018.

Financing a project is a painful question for us as well. Russian foundatians are willingly provide finance for the launch of the project but not for its support and development. At present, the attraction of sponsor funds has not been undertaken, since the project A. P. Ershov's Archive has been practically completed. The remaining digital projects of the Institute of Informatics Systems of the SB RAS are carried out within the framework of the government assignment to the Institute on the theme "Research of the fundamentals of data structuring, information resources management, creation of information and computing systems and environments for science and education" The purpose of this study is the development of automated support methods for ontology design. The bottleneck in this direction so far is the creation of more accurate search tools, text recognition tools, involving qualified personnel.

4 Technology and Method of Digital Historical Factography

As part of Internet-oriented professional IS's, the IIS SB RAS team has developed a technology and method of electronic historical factography, which allows working with arrays of heterogeneous documents and their further structuring by establishing connections between the entities reflected in the documents. The Internet resources developed at the IIS focus on the materials on the history of science and technology in Siberia – the Siberian Branch of the Russian (formerly Soviet) Academy of Sciences. Electronic historical factography dates back to 1999, when the IIS team began working on an automated information system for the creation and support of the Academician Andrey P. Ershov digital archive.

The method of electronic, or digital, historical factography consists in the publication of historical sources in Internet-oriented information systems according to the rules of working with conventional archival documents: properties such as the document source, type, author, addressee (either a person or an organization), dates, geographic data, etc., must be provided. The IS makes use of the technology allowing to establish connections between these entities of the subject field. Document quoting s from a digital archive is possible by means of web links as well as by indicating a specific volume and page of the archive (this is the case with the A. P. Ershov Electronic Archive and with other archives of the SB RAS where documents originate from the state-run storages).

While working on the first academic project of an Internet-oriented IS referred to as the A. P. Ershov Electronic Archive (http://ershov.iis.nsk.su), the IIS SB RAS team developed original software tools based on the client-server technology using predominantly Microsoft solutions. The archivist's workspace is written in Perl [7]. The specialized IS was created not only as a means of making a body of documents available to science, but also as a tool allowing a historian of science to perform a range of research tasks, such as organizing historical documents, providing remote access to these documents, keyword-based queries search, accumulating thematically connected sources from different storages, scientific description, etc. Almost all the documents from the Ershov's archive were digitalized with the exception of some personal letters.

The software tools created at the IIS SB RAS ensure stable functioning and continuous maintenance of virtual content. The developers wanted the visual archive in the public interface to correspond to the physical archive created by A. P. Ershov. He formed the cases on the thematic-chronological and thematic principles. His approach remained almost unchanged. Some corrections were made in order to remove duplicates, build chronology, establish authorship and dating of documents. The archive, formed by A. P. Ershov, was supplemented by some new documents received from the state archives. The electronic version supports two types of systematization: on the basis of cases and on the basis of thematic-chronological approach in the form of a corresponding catalog. Document types that were digitalized included letters, drafts of scientific arti-

cles, photos, reports, diaries, paperwork documents, reviews on scientific works, etc.

The Electronic Archive framework also contatins documents on the history of IIS SB RAS, VTNK "Start" (Temporary scientific and technical team "Start") and the A.P. Ershov Informatics Conference (PSI'). The pupils of the corresponding member of the USSR Ac. of Sci. S. S. Lavrov (1923-2004, St. Petersburg), transferred his archive to Novosibirsk. It is also presented on the platform of Ershov's archive.

The use of digitized documents is of communicational as well as of ergonomical importance, since many researchers with year of hands-on experience with archives suffer from the chronic disease caused by long-term contact with old paper, glue and dust, which at times prevents them from working directly in the archives. From this point of view, digital archives are safe and convenient to use. During the existence of the archive in the public domain, we did not receive objections to the publication of any documents.

5 Expansion of Project Activities

Upon the completion of the A. P. Ershov Electronic Archive project, in the run-up to the 50th Anniversary of the Siberian Branch of the RAS, an initiative group from the IIS SB RAS led by Dr. Alexander Marchuk began working on a new project – the SB RAS Electronic Photo Archive (http://www.soran1957.ru) (2005–2009). The project united a large number of separate photograph collections on the history of science in Siberia into a single volume of documents, which came from photoreporters, organizations (such as the SB RAS Museum, SB RAS Expo Center, SB RAS Press Center as well as a number of research institutes), and private collections. A landmark event in the history of Novosibirsk of the 20th century was setting up a town of science: Akademgorodok of Novosibirsk. We owe the existence photographic records of Akademgorodok from the moment of searching for a location for the new town to the foresight of Mikhail Lavrentiev, its founding father, who ordered that a cinema and photolaboratory be organized and invited Rashid Akhmerov (1926–2017) to be the staff photographer.

Especially for the Photo Archive a new IS was created– the SORAN1957 system [8]. It supports collecting, structuring and digital publication of historical data and documents using specially developed software and organizational mechanisms. The SORAN1957 includes a system of structured data reflecting real-world entities and their relationships. Methodologically, the system is based on the ideology of Semantic Web. This approach allows structuring data according to an ontology. An ontology is a formal specification of a shared conceptual model – an abstract model of the subject area describing a system of its concepts. A shared model is a conventional understanding of a conceptual model by a certain community (a group of people). "Specification" is an explicit description of the system of concepts, and "formal" means that the conceptual model is machine-readable. An ontology consists of classes of entities of a subject area,

properties of these classes, connections between these classes and statements made up of classes, their properties and connections. The resulting software tools enable input and editing of data as well as import of data from other sources, such as newspapers.

The SORAN1957 system features a public interface to the database and to the photographic documents. Users can study photographs, documents and database facts and their interconnections. For instance, by using text search the user can find a person of interest and their personal data, linked photographs, organizations (for instance, where that person worked), titles, etc. A nonspecific information ontology is used here, which allows avoiding the duplication of information contents in general-purpose and specialized information systems. The system is based on the Semantic Web concept and .NET technologies and can be installed on a server or an end user machine.

Our experience with the projects described above allowed us to cover a broader range of historical sources. In 2012, an integrative project of the SB RAS Presidium Fundamental Research, "The SB RAS Open Archive as a system of presenting, accumulation and systematization of scientific heritage" began (2012–2014, http://odasib.ru/). In this project, IIS SB RAS collaborated with a number of research institutes of the Siberian Branch specializing in humanities. Currently the SB RAS Open Archive contains 17 collections with 54,362 document scans (as of March 28, 2018). Documents added to each collections are systematized based on the internal logics of the content type. The system allows the creation of topic-based collections and sub-collections containing linked sources.

6 "Migration" Policy

It follows from the above descriptions of the projects that for each of them an original information system was developed, supported by grants from funds and sponsors (proprietary software). Some experts predicted over a dozen years ago that "in the future applied programs might not be developed but 'assembled' from ready-made components, a job that will not require a programmer but a qualified user who can formulate what he/she wants to receive at the output in the terms understood by the component management system. The center of gravity will shift from programming to design" [9]. Real life, however, has turned out to be much more complicated, and the key word here is a "qualified user."

In 2001, the open-source software expanded with the Drupal content management system (https://www.drupal.org/), developed and supported by programmers from all over the world [10]. The Drupal architecture allows for the construction of various web-applications like blogs, news sites, archives and social nets. Drupal contains over 40,000 modules that can be used to create an application necessary for solving the developers' problems. To achieve this, however, the developers need to learn how to find and install the necessary modules and how to write their own modules to solve highly specialized tasks. This means

that using information technologies by humanities-minded people is not a trivial task, and help from programmers is welcome if not a must.

In 2016, the A.P. Ershov Electronic Archive was migrated to the Drupal platform (the graduate project of Sergey N. Troshkov, Mechanics and Mathematics Department, Novosibirsk State University, supervised by Doctor of Physics and Mathematics Alexander G. Marchuk and programmer Marina Ya. Philippova) [11, **?**]. Parenthetically, following the migration of the Electronic Archive was the migration of the Library system developed by Ya. M. Kourliandchik for the A.P. Ershov Programming Department in the mid-1980s. Until recently, the latter system had been used by the IIS SB RAS but as it was not written in the client-server architecture and both the database and application were on installed the same computer, it could not be accessed from another computer [13].

The experience of developing the IS has revealed two approaches to project work. The approach to the creation of the A. P. Ershov's Electronic Archive is engineering: its creators used quite complex tools. Nevertheless, they have created a convenient and multifunctional system in the service and user interfaces. Achieving a workable version was a one-step process and did not require significant additions to the working tools. Changes and additions to the system architecture were made imperceptibly for operators and users, eliminating the loss or duplication of data. The tools were improved in the direction of increasing the speed of access to the database. All the developers of this system are currently the leading specialists of foreign software companies.

The approach of the creators of IS SORAN1957 and Open archive SB RAS can be called as researching. The system was developed with the help of complex Semantic Web tools. At the same time there was a search for the most optimal solutions in the creation of software. Variants of platform solutions have repeatedly changed, which sometimes led to duplication and loss of information, slowed down the work of operators, for some time stopped the filling of IS. The creators of the IS "Open archive SB RAS" did not provide short links to scans of documents.

7 Conclusion

An important scientific problem of electronic archives is the reliability of content. Professional historians believe that the researcher needs to see the original document in order to get the most complete picture of it. But the existing archiving system cannot provide a wide coverage of valuable archives. The creation of professional IS involves the responsibility of its developers for the quality of reproduction of documents. Modern means of representation allow the researcher to get enough information about the source. It is no coincidence that facsimiles and scans of rare books are being actively published.

Since the mid-1980s, the European community has launched projects supporting specialists engaged in the preservation, conservation and dissemination of knowledge about the heritage with the help of digital reality: Framework Programme for Research & Technological Development FR1, 1984–1987, continued

until 2013, and then HORIZON 2020 became its successor [14]. In addition to the programmes supporting appropriate research, special-purpose centers were set up in some countries like the U.K. and France to provide the long-term storage of software and access to it [15, 16]. Moreover, the European Commission is planning to launch a single European Open Science Cloud for storing, exchanging and reusing research data in a variety of areas and support its infrastructure. In Russia, apparently, the critical mass required for making such decisions at the national level has yet to be achieved. The Russian State Archives have begun publicizing their meetings and reference apparatus fairly recently, later than other institutions keeping historical sources. The Archive of the Russian Academy of Sciences (RAS) is the umbrella association for launching a universal corporate resource (http://www.isaran.ru). The Science Archive of the Siberian Branch, RAS, however, neither digitizes its collections nor is represented in the Internet. This is an urgent issue of the SB RAS and Russian Ministry for Science and Education.

The structural changes undergoing in the RAS Siberian Branch in connection with reforming the Russian Academy of Sciences have so far ignored the SB RAS archival activity. Therefore, the future of the SB RAS Science Archive is uncertain. This most valuable collection of documents on the development of Siberian science is in danger of neglect because the SB RAS Presidium has no funds to maintain or, more importantly, to develop it. The SB RAS Science Archive established simultaneously with the RAS Siberian Branch in 1958 possesses a richest array of representative sources on the history of science in Siberia. It includes 86 library collections and 52,219 files including 9,356 personal files. Until now, the Archive's library collections have not been digitized for professional or public purposes, and the Archive has no electronic resources of its own (even though the SB RAS State Public Scientific-Technical Library has the Internet connection). With a view to preserving the unique historical documents, we need to digitize them and establish permanent repositories of datasets using cloud technologies. Within the framework of the project SB RAS Open Archive, which is in line with the all-Russia trend for the extensive use of information and communication technologies in the cultural and scientific spheres, the IIS has pioneered the organization of archival work in the RAS Siberian Branch. We expect that our experience will be in demand.

8 Acknowledgements

Natalia Cheremnykh, Alexander Marchuk, Vladimir Philippov, Marina Philippova, Mikhail Bulyonkov, Andrey Nemov, Sergey Antuyfeev, Konstantin Fedorov, Irina Pavlovskaya, Alexander Rar, Natalia Poluydova, Igor Agamirzian, Ivan Golosov, Irina Adrianova. The study was carried out with the financial support of the Russian Foundation for Basic Research and the Novosibirsk Region, project №19-49-540001.

References

1. Lynch C. : Jim Gray's fourth paradigm and the construction of the scientific record. The Fourth Paradigm: Data-Intensive Scientific Discovery. T. Hey, S. Tansley, K. Tolle (eds.), pp. 175–182. Redmond, Washington, Microsoft Research (2009)
2. Mirskaya E.Z. : New information technologies in Russian science: history, results, problems and prospects. Science research : coll. Proc. A.I. Rakhitov (ed.). Moscow, INION RAN, pp. 174–200. (2011)
3. Krayneva I.A.: Electronic Archives on the History of Science. Vestnik NSU Series: History, Philology 12 (1), 76–83. (2013)
4. Deit Chrictofer. J. An Introduction to Database System. M. : Dialektika, 1998. 1070 p.
5. ISO/IEC 2382:2015 Information technology – Vocabulary: Information system – An information processing system, together with associated organizational resources such as human, technical, and financial resources, that provides and distributes information. http://www.morepc.ru/informatisation/iso2381-1.html#s
6. Gagarina D.A, Kiryanov I.K., Kornienko S.I. : History-oriented information systems: "Perm" project experience. Perm University Herald. History (16), 35 (2011).
7. Srinivasan S. Advanced Perl Programming. O'Reily Media Inc. 1997. 404 p.
8. Marchuk A.G., Marchuk P.A. : Archival factographic system. Digital Libraries: Advanced Methods and Technologies, Digital Collections. In : Proceedings of the XI All-Russian Scientific Conference (RCDL–2009), pp. 177–185 (2009)
9. Evtushkin A. : Dialectics and life of information technology. Computerra, 21 aug., 31 Available at: http://old.computerra.ru/197835/ (2001)
10. Mercer D. Drupal 7. Birmingham-Mumbai: Packt Publishing. 2010. 403 p.
11. James T. Migration to Drupal 7. Birmingham-Mumbai: Packt Publishing. 2012. 145 p.
12. Troshkov S.N. : Migrating Web Applications to the Freely Distributable Open Source Software. Bachelor's final qualifying work. Novosibirsk, NSU, 25 p. Scientific adviser M.Y. Fillipova, programmer IIS SB RAS. Authors archive (2016)
13. Troshkov S.N. : On Expirience in Migrating Applications to the Freely Distributable Open Source Software. Vestnik NSU Series: Information Technologies, 16 (2), 86–94. (2018)
14. Digital Heritage. Progress in Cultural Heritage: Documentation, preservation and protection. 2016. Nicosia, Cyprus, Oct. 31–Nov.5. In : Proceedings6th International conference, EuroMed, Part II. LNCS, vol. 10058, pp. V–VII (2016)
15. Doorn-Moiseenko T.L. : Electronic Archives and Their Role in the Development of the Information Infrastructure of Historical Science. In : Vorontsova, E.A., Aiani, V.Yu., Petrov, Yu.A.(eds.) Role of Archives in Information Support of Historical Science: a collection of articles, pp. 101–117. Moscow, ETERNA (2017)
16. Schurer K., Anderson S.J. with the assistance of Duncan J.A. : A Guide to Hictorical Datafiles Held in Machine-Readable Form. Assocoation for History and Computing. Cambridge, 339 p. http://www.aik-sng.ru/text/bullet/8/89-95.pdf (1992)

A Logical Approach to the Analysis of Aerospace Images

Valeriy Kuchuganov, Denis Kasimov, Aleksandr Kuchuganov

Kalashnikov Izhevsk State Technical University, Izhevsk, Russian Federation

kuchuganov@istu.ru, kasden@mail.ru, Aleks_KAV@udm.ru

Abstract. The paper proposes algorithms and software tools for the automatic interpretation and classification of objects and situations on aerospace images by structural-spatial analysis and iterative reasoning based on fuzzy logic and expert rules of inference. During iterations, the decision tree is built, the transition to local rules and additional features is carried out, and the ranges of acceptable values are adjusted. Particular attention is paid to geometric features of objects. Quantitative attributes are converted to qualitative ones for ease of perception of results and forming decision rules. The results of the experiment on the automatic identification of objects in the aerial image of an urban area are given. The system is useful for automating the process of labeling images for supervised learning and testing programs that recognize objects in aerospace images.

Keywords: Image Analysis, Object Detection, Object Features, Decision Rule, Decision Tree, Ground Truth, Image Labeling

1 Introduction

Due to the rapid growth in the volume of aerospace monitoring data, there is an urgent need for the tools that automate the extraction of knowledge from images, the identification and structured description of image objects.

In [1] the language for the description of (deformable) logical models of image objects is proposed, which is based on the PROLOG III language. The language is quite simple and transparent due to the use of the built-in predicates "line", "circle" and "texture", which are unified using a set of functions that dynamically extract low-level features from the image. The list of such functions includes Harris angle detector, LBP operator, support vector machine, spatial functions of the GEOS library, etc. The logical model of an object is a set of rules (statements). The found object satisfying the logical model is estimated by the function of energy that takes fuzzy values. The work of the interpreter is based on the classical search with returns and cutting off unpromising branches of the logical inference by checking the spatial constraints, using the A* algorithm and other popular strategies. The described approach is promising, but for a wide practical application there is a need to expand the set of built-in predicates and feature extraction functions, as well as significantly increase the speed of the object detection process.

Currently, the GEOBIA (Geographic Object-Based Image Analysis) approach [2] is actively being developed in the field of automation of aerospace image analysis. Within this approach, the processing of areas obtained as a result of automatic color segmentation and their classification based on rules set by an expert is implemented. There is an extensive experience of using the object-oriented approach for solving various tasks: analysis of changes in territories [3], classification of urban garden surfaces [4], automatic detection of built-up areas [5], estimation of crop residues [6], etc.

The work [7] is interesting by the proposed method of obtaining the object classification rules. The rules are formed on the basis of supervised learning: using manually prepared training examples, an automatic synthesis of a decision tree is performed, from which the most reliable classification rules are then manually extracted. The process of classifying objects in the image includes color image segmentation, calculation of spectral and geometric characteristics of the obtained segments, and classifying the segments into the target object categories by checking the rules that test feature values. In general, the considered work is aimed at maximizing the effectiveness of the final stage of image analysis, which is associated with decision making. It should be noted that the overall effectiveness of the analyzing system is determined not only by this factor, but also largely depends on the quality of image segmentation, the completeness of the set of analyzed features and the degree of consideration of the environment of objects.

In [8], an ontological approach to representing the knowledge of GEOBIA-systems is proposed. Based on the formalism of description logics, the relationships between the target categories of objects and their patterns in images are described. From these logical descriptions, it is then easy to extract the rules for assigning image objects provided by the segmentation procedure to the desired categories. The advantage of this approach is that the knowledge of experts is transferred to the analyzing system in a more systematic way, the subjectivity of the decision rules is leveled, and the possibility of using automatic means for checking the consistency of the knowledge base appears. Unfortunately, the approach does not specify any form of contextual analysis of objects. Identification relies entirely on spectral and geometric characteristics of individual objects (NDVI, squareness, etc.). The authors noted that automatic segmentation did not always perfectly delineate the boundaries of objects, especially in the case of shadows and trees. In this regard, the existing free database of cartographic data was used to refine the results of segmentation.

In existing implementations of the GEOBIA approach, relatively simple classification rules are applied that do not take into account the context of an object. The decision making mechanism is built on the production knowledge model. A significant drawback is the lack of tools for complex analysis of the shape of objects. Under conditions of imperfection of automatic image segmentation, it is not always possible to achieve high identification rates.

The purpose of this work is to develop mechanisms for the automatic interpretation and classification of objects and situations on aerospace images by structural-spatial analysis and iterative reasoning based on fuzzy logic and expert rules of inference. During iterations, the decision tree is built, the transition to local rules and additional

features is carried out, and the ranges of acceptable values are adjusted. Particular attention is paid to geometric features of objects. Quantitative attributes are converted to qualitative ones for ease of perception of results and forming decision rules.

2 Formation of a set of features

At the stage of image preprocessing, color segmentation and approximation of the edges of regions by circular arcs and straight line segments are performed. The algorithms of image approximation and extraction of basic features used by us are discussed in detail in [9]. The result of the stage is the set of ***color regions*** represented as the cyclic lists of straight line segments and circular arcs:

$$REGIONS = \{R_1, \ldots, R_n\},\ n \in \mathbf{N},\ R_i = (Color_i, Edge_i),\ Edge_i = (e_{i1}, \ldots, e_{ik}),\ k \in \mathbf{N},$$
$$\forall j \in [1..k]\ LineSegment(e_{ij}) \vee CircularArc(e_{ij}),\ Connected(e_{ik}, e_{i1}),\ \forall j \in [1..k-1]\ Connected(e_{ij}, e_{ij+1}),$$

where $Color_i$ is the region's average color; *LineSegment(e)* is true if e is a straight line segment; *CircularArc(e)* is true if e is a circular arc; $Connected(e_1, e_2)$ is true if any endpoints of e_1 and e_2 are the same.

Many of the obtained regions correspond unequivocally to the target objects of the image or their structural fragments (a typical example is a roof slope of a building). On the other hand, it is not possible to avoid regions that are incorrect to some degree: a very tortuous border, the capture of a part of a neighboring object, etc. In a number of researches [10, 11], it is noted that the inaccuracy, insufficiency or excessiveness of automatically obtained color segments is a serious problem for object-oriented approaches, preventing them from achieving a high level of recognition. In view of this, the subsequent stages of processing and analysis have been designed in such a way as to minimize the influence of this negative factor.

Next, ***the adjacency graph*** of the regions and sections of their edges is constructed. Each node of the graph corresponds to a certain region of the image. Arcs of the graph represent relationships between the regions:

$$(BegNode, EndNode, R, V, AdjChains),$$

where *BegNode* is the number of the node from which the arc exits; *EndNode* is the number of the node to which the arc enters; R is the type of relationship between the regions: *IsNeighbourOf*, *Contains*, *IsInsideOf*; V is the vector that connects the regions' centroids, showing relative orientation and distance; *AdjChains* is the set of adjacent chains of the regions' edges.

To describe the image regions, the following ***features*** are calculated:

1. Significant elements of the region's edge:

$$SignifEls_i = \{e \mid e \in Edge_i \wedge L(e) / P(Edge_i) \geq \delta\},$$

where L is the length calculation function, $L(e) \in \mathbf{R}^+$, P is the perimeter calculation function, $P(Edge_i) \in \mathbf{R}^+$, δ is a threshold, $\delta \in [0, 1]$.

2. Significant line segments: $SL_i = \{e \mid e \in SignifEls_i \wedge LineSegment(e)\}$.
3. Straightness of the region's edge:

$$Straightness(R_i) = \frac{\sum_{e \in SL_i} L(e)}{P(Edge_i)}.$$

4. The presence of three sides of a rectangle:

$$\exists a,b,c \in SL_i (a \mathrel{\tilde{\perp}} b, b \mathrel{\tilde{\perp}} c, a \mathrel{\tilde{\parallel}} c, Near(a,b), Near(b,c)) \rightarrow Has3RectSides(Edge_i)$$

where *Near* is true if the elements are located relatively close to each other; $\tilde{\perp}$ and $\tilde{\parallel}$ are the relations of fuzzy perpendicularity and parallelism that allow a slight deviation of the angle from 90° and 0°, respectively.

5. The presence of significant perpendicular line segments (*Has2PerpLines*).
6. *Area*, converted to a relative form through clustering.
7. Average width (*AvgWidth*), calculated on the basis of creating cross sections and clustering their lengths. The calculated absolute value is converted to a relative form, namely, it is considered depending on the size of the region.
8. Elongation of the region: $Elongation(R_i) = \min(a, b) / \max(a, b)$,
where a and b are the lengths of the sides of $MinBoundRect(R_i)$ – the minimum rectangle that covers the region R_i.
9. Squareness of the region: $Squareness(R_i) = Area(R_i) / Area(MinBoundRect(R_i))$,
where *Area* is the area calculation function, $Area(R_i) \in \mathbf{R}^+$.
10. *Circleness* – the ratio of the region's area to the area of a circle of the corresponding radius.
11. Tortuosity of the region's edge:

$$Tortuosity(R_i) = SignChangeCount(Edge_i) / P(Edge_i),$$

where *SignChangeCount* is the number of changes of the sign of the element inclination angle when traversing the edge.
12. Density of contour points: $ContourPointDensity(R_i) = |ContourPoints(R_i)| / Area(R_i)$,
where *ContourPoints* is the function that detects contour points in the given region of the image.

The last feature characterizes the texture of the region: if the value is small, then the region is homogeneous and smooth, otherwise it has a complex texture.

It should be noted that values of all features are relative. This ensures their invariance to different types of terrain and shooting conditions.

Quantitative values of the features are translated into qualitative form in order to simplify the process of forming decision rules and perception of the analysis results. For most features, it is convenient to operate with values from the following list: *Small*, *Medium*, *Large*, *VeryLarge*. For the *Color* feature, it is advisable to use the following values: *Green*, *YellowedGrassColor*, *Dark*, *VeryDark*, etc. Conversion of

values from the quantitative form to the qualitative one can be based on a simple partition of a value range into several subranges. However, a more flexible approach is to assign membership functions in accordance with the principles of fuzzy sets [12]. The second method is more laborious and time-consuming to set up, but the costs pay off to some extent, since some of classification errors associated with a slight violation of range boundaries are eliminated. At present, approaches are being developed [13, 14], aimed at simplifying the fuzzification process, seeking to eliminate subjectivity and reduce the share of manual labor in the creation of membership functions.

To facilitate the user's process of determining the ranges of qualitative values, the system has a tool for constructing histograms of the distribution of numerical values of the features. Fig. 1 shows a program window in which the *Straightness* feature is examined.

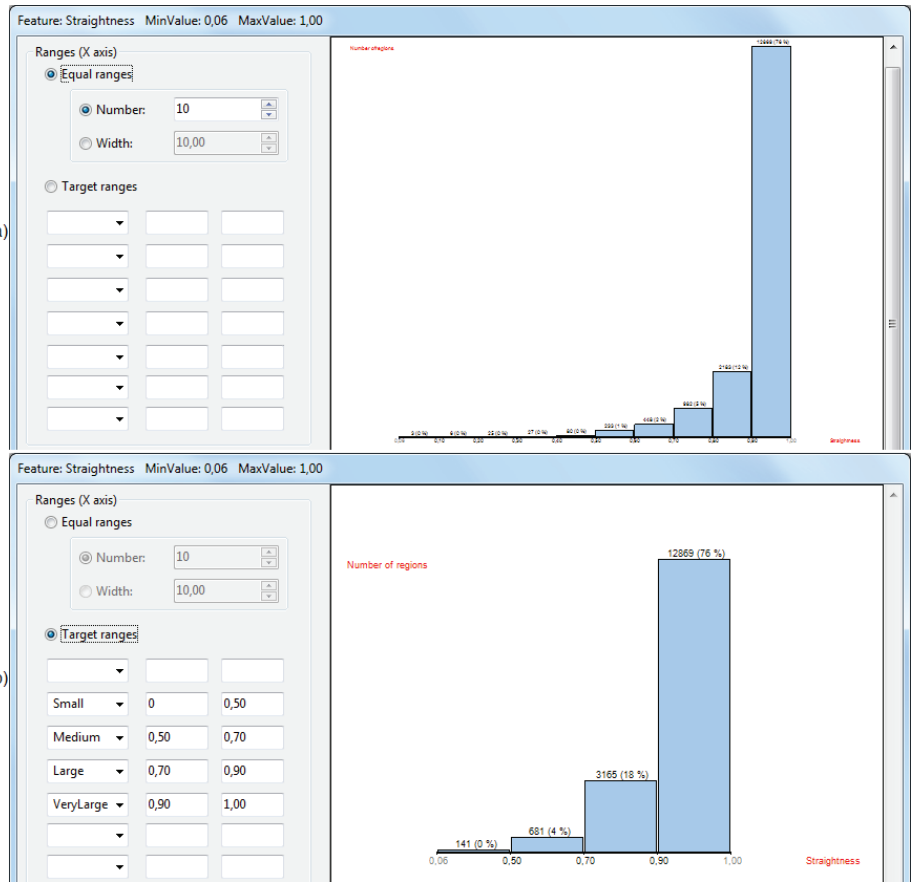


Fig. 1. Interface for converting quantitative values of the features into qualitative ones: (a) the initial equal ranges; (b) the target ranges *Small*, *Medium*, *Large*, and *VeryLarge*

Initially, the histogram is built with a large number of equal ranges (Fig. 1-a) to give the user a general idea of the shape of the distribution. The task of the user is to reduce the number of ranges to the required number of qualitative values of the feature. For example, in Fig. 1-b, the user has defined ranges of four qualitative straightness values that he considers sufficient for classification.

By clicking on any bar of the histogram, the system highlights all the color areas in the image for which the feature value falls within the range corresponding to this bar. This feature helps the user to evaluate the correctness of the resulting ranges.

The qualitative values of the features determined in the manner described are then used to specify the decision rules. During the interpretation of the rules, the quantitative values are fuzzified using simple built-in trapezoidal functions, which are automatically scaled to the width of the user-defined ranges.

3 Formation of decision rules

Decision rules are divided into the following types:

- rules that analyze image regions in isolation from other regions, classifying the most reliable objects;
- rules that consider aggregates of adjacent regions, classifying less reliable objects and refining previously detected objects.

Consider some rules of the first type, applied at the beginning of the object classification stage:

Object Category: Building.

General requirements: $(Color(R) \neq Green) \wedge (Color(R) \neq VeryDark) \wedge (Area(R) \geq Medium) \wedge (Elongation(R) \leq Medium) \wedge (AvgWidth(R) \geq VeryLarge) \wedge (ContourPointDensity(R) \leq Medium) \wedge (\neg \exists R_2\ (Color(R_2) = Green \wedge Contains(R, R_2))$.

Variants:

- **IF** $(Straightness(R) \geq VeryLarge)$ **THEN** $Building(R, 1.0)$.
- **IF** $(Squareness(R) \geq Large) \wedge (Has3RectSides(R) \vee Has2PerpLines(R))$ **THEN** $Building(R, 1.0)$.
- **IF** $(Straightness(R) \geq Large) \wedge (Has3RectSides(R) \vee Has2PerpLines(R)) \wedge (Squareness(R) \geq Medium) \wedge (Tortuosity(R) \leq Medium)$ **THEN** $Building(R, 1.0)$.
- **IF** $(Straightness(R) \geq Large) \wedge (Has3RectSides(R) \vee Has2PerpLines(R)) \wedge (Tortuosity(R) \leq Medium)$ **THEN** $Building(R, 0.7)$.
- **IF** $(Squareness(R) \geq Large)$ **THEN** $Building(R, 0.5)$.
- **IF** $(\exists e \in R.Edge\ L(e) \geq VeryLarge)$ **THEN** $Building(R, 0.5)$.

Object Category: Shadow of Building.

General requirements: $(Color(R) = VeryDark)$.

Variants:

- **IF** $(Straightness(R) \geq VeryLarge)$ **THEN** $ShadowOfBuilding(R, 1.0)$.

— **IF** (*Straightness*(*R*) ≥ *Large*) **THEN** *ShadowOfBuilding*(*R*, 0.8).
— **IF** (*Straightness*(*R*) ≥ *Medium*) ∧ (*Tortuosity*(*R*) ≤ *Large*) **THEN** *ShadowOfBuilding*(*R*, 0.6).

When a rule is triggered, the image region obtains a classification variant with the degree of reliability specified in the rule's consequent. The reliability value is set by the expert.

The rules of the second type are repeatedly applied to the results of the previous classification steps, while new information is added. Below is an example of one of these rules:

Object Category: Building near ShadowOfBuilding.

IF *ShadowOfBuilding*(R_2, *m*), *Adjacent*(*R*, R_2) ∧ (*Straightness*(*CommonEdge*(*R*, R_2)) ≥ *Large*) ∧ (*Orientation*(R_2, *R*) ≅ *SolarAngle*) ∧ … **THEN** *Building*(*R*, *m*), *ShadowOfBuilding*(R_2, *m* + 0.2),

where *m* is the reliability of classification of the region R_2 before executing the rule.

The classification stage has an iterative principle of organization. Different sets of rules can be used at different iterations. Namely, when setting rules, the expert can specify on which iterations they can be applied. This principle makes it possible to implement various image analysis strategies. For example, the rules used in the *i*-th iteration can serve the purpose of detecting all potential objects, ensuring maximum recall rate and not worrying much about the precision. Then the elimination of false objects can be made on the *i+1* iteration by specifying additional features, checking the contextual rules and adjusting the ranges of acceptable values.

In our experiments, iterations were used as follows: (1) detection of the most reliable objects; (2) classification of less reliable objects with the condition that they are adjacent to reliable objects; (3) identifying buildings near shadows the source of which has not yet determined; (4) detection of buildings and roads among the remaining regions on the basis of weakened requirements; (5) classifying the remaining regions into the categories of trees, grass, and roads (what they are more like, depending on tortuosity and color).

The results of rule execution are organized as a decision tree, the general structure of which is shown in Fig. 2. Each color image region has its own decision tree. Of all the classifications derived, the one with the highest reliability is chosen as the result. Similarly, when interpreting rules of the second type, the neighbors are substituted in descending order of the degree of compliance with the specified requirements.

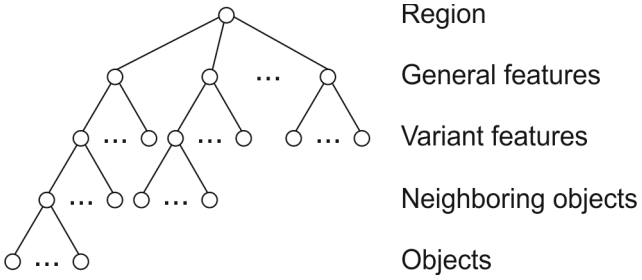


Fig. 2. Decision tree structure

4 Experiment

An experiment on the automatic classification of objects was conducted on an urban area aerial image taken from Inria Aerial Image Labeling Dataset [15]. Fig. 3 shows examples of the work of our approach and two other approaches in the literature.

Fig. 3. A visual comparison of the results: (a) the original image; (b) FCN results [15]; (c) MLP results [15]; (d) our results

The dataset [15] contains the building ground truth, providing an opportunity to assess the quality of building identification using the *Intersection over Union* (*IoU*) and *Accuracy* [16, 17] metrics. These measures are calculated as follows:

$$IoU = |A \cap G| / |A \cup G|, \; Accuracy = |A \cap G| / |A|,$$

where A is the set of pixels that the program has classified as pixels of target objects; G is the set of pixels that are related to target objects in the ground truth.

As a result of automatic analysis of the image that has the size 5000×5000 and contains a total of 793 buildings, the following performance values were obtained:

$$IoU = 0.56, \; Accuracy = 0.83.$$

The values obtained can be considered satisfactory. It should be noted that the applied metrics work at the pixel level and require the most accurate determination of

the boundaries of buildings. In this paper, the desire for accurate detection of objects was not put at the forefront. The most difficult to classify were small buildings partially covered with trees, since their visible parts often do not have any distinctive elements of geometric shape. Such situations require special analysis strategies. Neural network approaches [15] also experience some difficulties on this dataset (the average value of *IoU* does not exceed 0.6), leaving considerable room for improvement.

Practice shows that manual creation of the ground truth labeling for a single large aerospace image takes more than an hour. And additionally it is necessary to prepare a set of test images. The obtained estimate *IoU*=56% indicates the possibility of reducing labor costs by half. The important point is that the developed system does not require training and can be relatively easily reconfigured to other images. Thus, the system can be useful for automating the creation of training datasets to expand the scope of application of artificial neural networks.

5 Conclusion

Thus, the proposed approach to the analysis of aerospace images is based on structural-spatial analysis and iterative reasoning with the use of fuzzy logic and expert rules of inference. During iterations, the decision tree is built, the transition to local (contextual) rules and additional features is carried out, and the ranges of acceptable values are adjusted.

Particular attention is paid to geometric features of objects. The set of standard geometric characteristics (perimeter, area, squareness, circleness, elongation, etc.) of objects has been supplemented with such more complex features as significant elements, straightness, presence of three sides of a rectangle, presence of significant perpendicular line segments, tortuosity, average width, and contour point density.

The advantages of the logical approach to image analysis are the following: (a) argumentation of the decision; (b) the possibility of context-sensitive analysis; (c) automatic generation of descriptions of objects and scenes; (d) there is no need for training on labeled datasets; (e) relatively simple adjustment to the required type of images and shooting conditions.

Based on the classification results, it is possible to form a training dataset for neural network type recognition systems. This may require some manual adjustment of the results: removal of false objects and refinement of the edges of true objects. If necessary, two-stage training can be organized. In this case, at the second level, with the help of additional features, “specialization” is carried out according to the seasons (winter, summer, autumn), types of terrain (agricultural grounds, highland), and other parameters of shooting.

Further enhancement of the system is seen in the organization of flexible search strategies, for example, specific techniques for large/extended/small objects. Due to context-sensitive strategies, the system will automatically, depending on the content of a particular area, adapt to the shooting conditions, the texture of objects, combinations of objects of different categories, the nature of the edges between them, etc.

Acknowledgment

This work is supported by the Russian Science Foundation under grant No. 18-71-00109.

References

1. Bychkov, I.V., Ruzhnikov, G.M., Fedorov, R.K., Avramenko, Y.V.: Interpretator yazyka SOQL dlya obrabotki rastrovykh izobrazheniy [The interpreter of the SOQL language for processing raster images]. Vychislitel'nyye tekhnologii = Computing technologies, 21(1), 49–59 (2016). (in Russian).
2. Blaschke, T., Hay, G.J., Kelly, M., Lang, S., Hofmann, P., Addink, E., Feitosa, R.Q., Meer, F., Werff, H., Coillie, F., Tiede, D.: Geographic Object-Based Image Analysis – Towards a new paradigm. ISPRS Journal of Photogrammetry and Remote Sensing **87**, 180–191 (2014). doi: 10.1016/j.isprsjprs.2013.09.014
3. Souza-Filho, P.W.M., Nascimento, W.R., Santos, D.C., Weber, E.J., Silva, R.O., Siqueira, J.O.: A GEOBIA Approach for Multitemporal Land-Cover and Land-Use Change Analysis in a Tropical Watershed in the Southeastern Amazon. Remote Sensing **10**(11), 1683 (2018). doi: 10.3390/rs10111683
4. Baker, F., Smith, C.: A GIS and object based image analysis approach to mapping the greenspace composition of domestic gardens in Leicester, UK. Landscape and Urban Planning **183**, 133–146 (2019). doi: 10.1016/j.landurbplan.2018.12.002
5. Lehner, A., Naeimi, V., Steinnocher, K.: Sentinel-1 for object-based delineation of built-up land within urban areas. In: Ragia L. et al. (Eds.). Geographical Information Systems Theory, Applications and Management. Third International Conference, GISTAM 2017. Communications in Computer and Information Science, vol. 936, pp.1–18 (2019). doi: 10.1007/978-3-030-06010-7_2
6. Najafi, P., Navid, H., Feizizadeh, B., Eskandari, I.: Object-based satellite image analysis applied for crop residue estimating using Landsat OLI imagery. International Journal of Remote Sensing **39**(19), 6117–6136 (2018). doi: 10.1080/01431161.2018.1454621
7. Antunes, R.R., Bias, E.S., Costa, G.A.O.P., Brites, R.S.: Object-Based Analysis For Urban Land Cover Mapping Using The InterIMAGE And The SIPINA Free Software Packages. Bulletin of Geodetic Sciences **24**(1), 1–17 (2018). doi: 10.1590/s1982-21702018000100001
8. Belgiu, M., Hofer, B., Hofmann, P.: Coupling formalized knowledge bases with object-based image analysis. Remote Sensing Letters **5**(6), 530–538 (2014). doi: 10.1080/2150704X.2014.930563
9. Kasimov, D.R., Kuchuganov, A.V., Kuchuganov, V.N., Oskolkov, P.P.: Approximation of Color Images Based on the Clusterization of the Color Palette and Smoothing Boundaries by Splines and Arcs. Programming and Computer Software **44**(5), 295–302 (2018). doi: 10.1134/S0361768818050043
10. Lhomme, S., He, D.C., Weber, C., Morin, D.: A new approach to building identification from very-high-spatial-resolution images. Int. J. Remote Sens. **30**, 1341–1354 (2009). doi: 10.1080/01431160802509017
11. You, Y, Wang, S., Ma, Y., Chen, G., Wang, B., Shen, M., Liu, W.: Building Detection from VHR Remote Sensing Imagery Based on the Morphological Building Index. Remote Sensing **10**(8), 1288 (2018). doi: 10.3390/rs10081287

12. Zadeh, L.A.: The concept of a linguistic variable and its application to approximate reasoning—I. Information Sciences **8**(3), 199–249 (1975). doi: 10.1016/0020-0255(75)90036-5
13. Liao, T.W.: A procedure for the generation of interval type-2 membership functions from data. Applied Soft Computing **52**, 925–936 (2017). doi: 10.1016/j.asoc.2016.09.034
14. Dhar, S., Kundu, M.K.: A novel method for image thresholding using interval type-2 fuzzy set and Bat algorithm. Applied Soft Computing **63**, 154–166 (2018). doi: 10.1016/j.asoc.2017.11.032
15. Maggiori, E., Tarabalka, Y., Charpiat, G., Alliez, P. Can Semantic Labeling Methods Generalize to Any City? The Inria Aerial Image Labeling Benchmark. IEEE International Geoscience and Remote Sensing Symposium, IGARSS 2017 (2017). doi: 10.1109/IGARSS.2017.8127684
16. Sokolova, M., Lapalme, G.: A systematic analysis of performance measures for classification tasks. Information Processing and Management **45**(4), 427–437 (2009). doi: 10.1016/j.ipm.2009.03.002
17. Fernandez-Moral, E., Martins, R., Wolf, D., Rives, P.: A new metric for evaluating semantic segmentation: leveraging global and contour accuracy. Workshop on Planning, Perception and Navigation for Intelligent Vehicles, PPNIV17 2017 (2017). doi: 10.1109/IVS.2018.8500497

Parallel Factorization of Boolean Polynomials*

Vadiraj Kulkarni[1], Pavel Emelyanov[2,3], Denis Ponomaryov[2,3], Madhava Krishna[1,4], Soumyendu Raha[1], and S K Nandy[1]

[1] Computer Aided Design Laboratory, Indian Institute of Science, Bangalore 560012
{vadirajk,madhava,raha,nandy}@iisc.ac.in
[2] Ershov Institute of Informatics Systems, Lavrentiev av. 6, 630090, Novosibirsk, Russia
[3] Novosibirsk State University, Pirogova st. 1, 630090, Novosibirsk, Russia
{emelyanov,ponom}@iis.nsk.su
[4] Morphing Machines Pvt. Ltd

Abstract. Polynomial factorization is a classical algorithmic problem in algebra, which has a wide range of applications. Of special interest is factorization over finite fields, among which the field of order two is probably the most important one due to the relationship to Boolean functions. In particular, factorization of Boolean polynomials corresponds to decomposition of Boolean functions given in the Algebraic Normal Form. It has been also shown that factorization provides a solution to decomposition of functions given in the full DNF (i.e., by a truth table), for positive DNFs, and for cartesian decomposition of relational datatables. These applications show the importance of developing fast and practical factorization algorithms. In the paper, we consider some recently proposed polynomial time factorization algorithms for Boolean polynomials and describe a parallel MIMD implementation thereof, which exploits both the task and data level parallelism. We report on an experimental evaluation, which has been conducted on logic circuit synthesis benchmarks and synthetic polynomials, and show that our implementation significantly improves the efficiency of factorization. Finally, we report on the performance benefits obtained from a parallel algorithm when executed on a massively parallel many core architecture (Redefine).

Keywords: Boolean Polynomials · Factorization · Reconfigurable Computing.

1 Introduction

Polynomial factorization is a classical algorithmic problem in algebra, [8], which has numerous important applications. An instance of this problem, which deserves a particular attention, is factorization of Boolean polynomials, i.e., multilinear polynomials over the finite field of order 2. A Boolean polynomial is one

* This work was supported by the grant of Russian Foundation for Basic Research No. 17-51-45125 and by the Ministry of Science and Education of the Russian Federation under the 5-100 Excellence Program.

of the well-known sum-of-product representations of Boolean functions known as Zhegalkine polynomials [14] in the mathematical logic or the Reed–Muller canonical form [10] in the circuit synthesis. The advantage of this form that has recently made it popular again is a more natural and compact representation of some classes of Boolean functions (e.g., arithmetical functions, coders/cyphers, etc.), a more natural mapping to some circuit technologies (FPGA–based and nanostructure–based electronics), and good testability properties.

Factorization of Boolean polynomials is a particular case of decomposition (so–called disjoint conjunctive or AND–decomposition) of Boolean functions. Indeed, in a Boolean polynomial each variable has degree at most 1, which makes the factors have disjoint variables: $F(X,Y) = F_1(X) \cdot F_2(Y)$, $X \cap Y = \varnothing$.

It has been recently shown [4, 5] that factorizaton of Boolean polynomials provides a solution to conjunctive decomposition of functions given in the full DNF (i.e., by a truth table) and for positive DNFs without the need of (inefficient) transformation between the representations. Besides, it provides a method for Cartesian decomposition of relational datatables [3, 6], i.e., finding tables such that their unordered Cartesian product gives the source table. We give some illustrating examples below.

Consider the following DNF

$$\varphi = (x \wedge u) \ \vee \ (x \wedge v) \ \vee \ (y \wedge u) \ \vee \ (y \wedge v) \vee (x \wedge u \wedge v)$$

It is equivalent to

$$\psi = (x \wedge u) \ \vee \ (x \wedge v) \ \vee \ (y \wedge u) \ \vee \ (y \wedge v)$$

since the last term in φ is redundant. One can see that

$$\psi \equiv (x \vee y) \wedge (u \vee v)$$

and the decomposition components $x \vee y$ and $u \vee v$ can be recovered from the factors of the polynomial

$$F_\psi = xu + xv + yu + yv = (x + y) \cdot (u + v)$$

constructed for ψ.

The following full DNF

$\varphi = (x \wedge \neg y \wedge u \wedge \neg v) \vee (x \wedge \neg y \wedge \neg u \wedge v) \vee$
$\vee (\neg x \wedge y \wedge u \wedge \neg v) \vee (\neg x \wedge y \wedge \neg u \wedge v)$

is equivalent to

$$(x \wedge \neg y) \vee (\neg x \wedge y) \bigwedge (u \wedge \neg v) \vee (\neg u \wedge v)$$

and the decomposition components of φ can be recovered from the factors of the polynomial

$$F_\varphi = x\bar{y}u\bar{v} + x\bar{y}\bar{u}v + \bar{x}yu\bar{v} + \bar{x}y\bar{u}v = (x\bar{y} + \bar{x}y) \cdot (u\bar{v} + \bar{u}v) \qquad (1)$$

constructed for φ.

Finally, Cartesian decomposition of the following table

B	E	D	A	C
z	q	u	x	y
y	q	u	x	y
y	r	v	x	z
z	r	v	x	z
y	p	u	x	x
z	p	u	x	x

$=$

A	B
x	y
x	z

$\times$

C	D	E
x	u	p
y	u	q
z	v	r

can be obtained from the factors of the polynomial

$$\begin{aligned} &z_B \cdot q \cdot u \cdot x_A \cdot y_C + \; y_B \cdot q \cdot u \cdot x_A \cdot y_C + \\ &y_B \cdot r \cdot v \cdot x_A \cdot z_C + \; z_B \cdot r \cdot v \cdot x_A \cdot z_C + \\ &y_B \cdot p \cdot u \cdot x_A \cdot x_C + \; z_B \cdot p \cdot u \cdot x_A \cdot x_C = \\ &\qquad = (x_A \cdot y_B + x_A \cdot z_B) \cdot (q \cdot u \cdot y_C + r \cdot v \cdot z_C + p \cdot u \cdot x_C) \end{aligned}$$

constructed for the table's content.

Decomposition facilitates finding a more compact representation of Boolean functions and data tables, which is applied in the scope of the Logic Circuit Synthesis, self-organizing databases, and dependency mining, respectively. Due to the typically large inputs in these tasks, it is important to develop efficient and practical factorization algorithms for Boolean polynomials.

In [13], Shpilka and Volkovich showed a connection between polynomial factorization and identity testing. It follows from their results that a Boolean polynomial can be factored in time $O(l^3)$, where l is the size of the polynomial given as a symbol sequence. The approach employs multiplication of polynomials obtained from the input one, which is a costly operation in case of large inputs. In [4], Emelyanov and Ponomaryov proposed an alternative approach to factorization and showed that it can be done without explicit multiplication of Boolean polynomials. The approach has been further discussed in [7].

In this paper, we propose a parallel version of the decomposition algorithm from [4, 7]. In Section 2, we revisit the sequential factorization algorithm from these papers. In Section 3, we describe a parallel MIMD implementation of the algorithm and further in Section 4 we perform a quantitative analysis of the parallel algorithm versus the sequential one. Finally, in Section 5 we evaluate our algorithm on a massively parallel many core architecture (Redefine) and outline the results.

2 Background

In this section we reproduce the sequential algorithm from [4, 7] for the ease of exposition. Let us first introduce basic definitions and notations.

A polynomial $F \in \mathbb{F}_2[x_1, \ldots, x_n]$ is called *factorable* if $F = F_1 \cdot \ldots \cdot F_k$, where $k \geq 2$ and $F_1, \ldots, F_k$ are non-constant polynomials. The polynomials $F_1, \ldots, F_k$ are called *factors* of F. It is important to realize that since we consider multilinear polynomials (every variable can occur only in the power of ≤ 1), the factors are polynomials *over disjoint sets of variables*. In the following sections,

we assume that the polynomial F does not have *trivial divisors*, i.e., neither x, nor $x+1$ divides F. Clearly, trivial divisors can easily be recognized.

For a polynomial F, a variable x from the set of variables $Var(F)$ of F, and a value $a \in \{0,1\}$, we denote by $F_{x=a}$ the polynomial obtained from F by substituting x with a. $\frac{\partial F}{\partial x}$ denotes a *formal derivative* of F wrt x. Given a variable z, we write $z|F$ if z divides F, i.e., z is present in every monomial of F (note that this is equivalent to the condition $\frac{\partial F}{\partial z} = F_{z=1}$). Given a set of variables Σ and a monomial m, the *projection* of m onto Σ is 1 if m does not contain any variable from Σ, or is equal to the monomial obtained from m by removing all the variables not contained in Σ, otherwise. The *projection* of a polynomial F onto Σ, denoted by $F|_\Sigma$, is the polynomial obtained as the sum of monomials from the set S projected onto Σ, with duplicate monomials removed.

2.1 Factorization Algorithm

Algorithm 1 describes the sequential version of the factorization algorithm. As already mentioned, the factors of a Boolean polynomial have disjoint sets of variables. This property is employed in the algorithm, which tries to compute a variable partition. Once it is computed, the corresponding factors can be easily obtained as projections of the input polynomial onto the sets from the partition.

The algorithm chooses a variable randomly from the variable set of F. Assuming the polynomial F contains at least two variables the algorithm partitions the variable set of F into two sets with respect to the chosen variable:

- the first set Σ_{same} contains the selected variable and corresponds to an irreducible polynomial;
- the second set Σ_{other} corresponds to the second polynomial which can admit further factorization.

The factors of F, F_{same} and F_{other} are obtained as the projections of the input polynomial onto Σ_{same} and Σ_{other}, respectively.

In lines 1-3, we select an arbitrary variable x from the variable set of F and compute the polynomials A and B. A is the derivative of F wrt x and B is the polynomial obtained by setting x to zero in F. In lines 4-10, we loop through the variable set of F excluding x, calculate the polynomials C and D, and check if the product AD is equal to BC. C is the derivative of polynomial A and D is the derivative of polynomial B. To check whether AD is equal to BC we invoke the **IsEqual** procedure in line 6. We describe the **IsEqual** procedure in detail in the next subsection.

2.2 IsEqual Procedure

Algorithm 2 describes the sequential version of the IsEqual procedure.

Algorithm 1 Sequential Factorization Algorithm

Input Boolean polynomial to be factored F
Output F_{same} and F_{other} which are the factors of the input polynomial F

1: Take an arbitrary variable x occurring in F
2: Let $A = \frac{\partial F}{\partial x}, B = F_{x=0}$
3: Let $\Sigma_{same} = x, \Sigma_{other} = \emptyset, F_{same} = 0, F_{other} = 0$
4: **for each** $y \in var(F) \setminus \{x\}$ **do**
5: Let $C = \frac{\partial A}{\partial y}$, $D = \frac{\partial B}{\partial y}$
6: **if** IsEqual(A, D, B, C) **then**
7: $\Sigma_{other} = \Sigma_{other} \cup \{y\}$
8: **else**
9: $\Sigma_{same} = \Sigma_{same} \cup \{y\}$
10: **end if**
11: **end for**
12: If $\Sigma_{other} = \emptyset$ then F is non-factorable
13: Return polynomials F_{same} and F_{other} obtained as projections onto Σ_{same} and Σ_{other} respectively.

- The procedure takes input polynomials A, B, C, D and computes whether $AD = BC$ by employing recursion.
- Lines 1-2,7-16 implement the base cases when $AD = BC$ can be determined trivially.
- In Line 3-5, we check whether a variable z divides the polynomials A, B, C, D such that the condition in Line 4 holds. If this is not the case, then we can eliminate z from A, B, C, D and check if the products of the resulting polynomials are equal.
- In Lines 17-25, we recursively invoke **IsEqual** procedure on polynomials, whose sizes are smaller than the size of the original ones.

2.3 Scope for Parallelism

The crux of Algorithm 1 is the loop in Lines 4-11. We observe that the different iterations of the loop are independent of each other. Hence the loop exhibits thread level parallelism which can be exploited for performance gain. The conditional block inside the loop in Lines 6-10 can be used to exploit the task level parallelism between the multiple threads.
Multiple sections of Algorithm 2 are amenable for parallelization. Checking the divisibility of the polynomials A, B, C, D in Lines 3-6 of **IsEqual** procedure can be performed independently. In Lines 16-23, the recursive calls to **IsEqual** procedure are independent of each other and exhibit thread level parallelism.
In the next section we propose a parallel algorithm using the above observations.

Algorithm 2 Sequential IsEqual Procedure

Input Boolean polynomials A,B,C,D
Output TRUE if AD is equal to BC and FALSE otherwise.

1: If A=0 or D=0 then return (B=0 or C=0)
2: If B=0 or C=0 then return FALSE
3: **for each** z occurring in at least one of A,B,C,D **do**
4: **if** $z|A$ or $z|D$ xor $z|B$ or $z|C$ **then**
5: return FALSE
6: **end if**
7: Replace every $X \in \{A, B, C, D\}$ with $\frac{\partial X}{\partial z}$, provided $z|X$
8: **end for**
9: **if** A=1 and D=1 **then** return (B=1 and C=1)
10: **end if**
11: **if** B=1 and C=1 **then** return FALSE
12: **end if**
13: **if** A=1 and B=1 **then** return (D=C)
14: **end if**
15: **if** D=1 and C=1 **then** return (A=B)
16: **end if**
17: Pick a variable z
18: **if** not(IsEqual($A_{z=0}, D_{z=0}, B_{z=0}, C_{z=0}$)) **then** return FALSE
19: **end if**
20: **if** not(IsEqual($\frac{\partial A}{\partial z}$, $\frac{\partial D}{\partial z}$, $\frac{\partial B}{\partial z}$, $\frac{\partial C}{\partial z}$)) **then** return FALSE
21: **end if**
22: **if** IsEqual($\frac{\partial A}{\partial z}, B_{z=0}, A_{z=0}, \frac{\partial B}{\partial z}$) **then** return TRUE
23: **end if**
24: **if** IsEqual($\frac{\partial A}{\partial z}, C_{z=0}, A_{z=0}, \frac{\partial C}{\partial z}$) **then** return TRUE
25: **else** return FALSE
26: **end if**

3 Proposed Approach

3.1 Parallel Factorization Algorithm

Algorithm 3 describes the parallel version of the factorization algorithm. In Lines 1-3, we select an arbitrary variable x from the variable set of F and compute the polynomials A and B. In Lines 4-11, we perform multiple loop iterations independently in parallel by spawning multiple threads. Each thread will return two sets Σ^{tid}_{same} and Σ^{tid}_{other} specific to the scope of the thread designated by thread identifier tid. In Lines 12-13, the variable sets Σ_{same} and Σ_{other} are computed as the union of the thread specific instances, respectively. Note that Lines 12-13 perform barrier synchronization of all the parallel threads.

3.2 Parallel IsEqual Procedure

Algorithm 4 describes the parallel version of the IsEqual procedure. This algorithm takes as input four polynomials A, D, B, C and checks whether the product AD is equal to the product BC. Lines 1-2 and lines 14-21 describe the cases

Algorithm 3 Parallel Decomposition Algorithm

Input Boolean polynomial to be factored F
Output F_{same} and F_{other} which are the factors of the input polynomial F

1: Take an arbitrary variable x occurring in F
2: Let $A = \frac{\partial F}{\partial z}, B = F_{z=0}$
3: Let $\Sigma_{same} = x, \Sigma_{other} = \emptyset, F_{same} = 0, F_{other} = 0$
4: **for each** $y \in var(F) \setminus \{x\}$ **do in parallel**
5: Let $C = \frac{\partial A}{\partial y}$ $D = \frac{\partial B}{\partial y}$
6: **if** IsEqual(A, D, B, C) **then**
7: $\Sigma^{tid}_{other} = \Sigma^{tid}_{other} \cup \{y\}$
8: **else**
9: $\Sigma^{tid}_{same} = \Sigma^{tid}_{same} \cup \{y\}$
10: **end if**
11: **end for Wait for all the parallel threads to finish**
12: $\Sigma_{other} = \bigcup_{tid} \Sigma^{tid}_{other}$
13: $\Sigma_{same} = \bigcup_{tid} \Sigma^{tid}_{same}$
14: If $\Sigma_{other} = \emptyset$ then F is non-factorable; stop
15: Return polynomials F_{same} and F_{other} obtained as projections onto Σ_{same} and Σ_{other}, respectively.

when determining $AD = BC$ is trivial. In lines 3-9, we check whether a variable z divides the input polynomials A, D, B, C such that the condition in Line 5 holds. If this is not the case, we divide them by z to obtain the reduced polynomials. The above operations are performed for each variable independently in parallel by spawning multiple threads. In Line 8 each thread checks whether a variable z^{tid} (tid denotes the thread id) is a divisor of any of A, B, C, D. If z^{tid} divides any of A, B, C, D it computes the corresponding reduced polynomials $A^{tid}, D^{tid}, B^{tid}, C^{tid}$ obtained by dividing any of A, D, B, C by z^{tid}, respectively. In line 10 we wait for all the threads to finish. In Line 13 we take pairwise intersection of the corresponding monomials of thread specific polynomials $A^{tid}, D^{tid}, B^{tid}, C^{tid}$ to form polynomials which are free of trivial divisors. Intersection of two monomials is a monomial containing the variables present in both. In Lines 23-27, we perform four recursive calls to the IsEqual function independently in parallel by spawning multiple threads. In Line 28-37, we wait for all the threads to finish and compare the outputs of each threads to form the final output. Note that lines 10 and 28 perform barrier synchronization of all the parallel threads.

4 Experiments and Results

Experimental evaluation of the sequential and parallel algorithms was made on Logic circuit synthesis benchmarks and synthetic Boolean polynomials.

Algorithm 4 Parallel IsEqual Function

Input Boolean polynomials A,B,C,D
Output TRUE if AD is equal to BC and FALSE otherwise.

If A =0 or D=0 then return (B=0 or C=0)
If B=0 or C=0 then return FALSE
for each z occurring in at least one of A,B,C,D **do in parallel**
    set $flag^{tid}$= True
    if $z|A$ or $z|D$ xor $z|B$ or $z|C$ **then**
        set $flag^{tid}$= FALSE
    end if
    Replace every $X^{tid} \in \{A, B, C, D\}$ with $\frac{\partial X^{tid}}{\partial z}$, provided $z|X^{tid}$
end for
Wait for all threads to finish
if $\bigwedge_{tid} flag^{tid} = FALSE$ **then** return FALSE
end if
$X = \bigcap_{tid} X^{tid}$, for $X \in \{A, B, C, D\}$
if A=1 and D=1 **then** return (B=1 and C=1)
end if
if B=1 and C=1 **then** return FALSE
end if
if A=1 and B=1 **then** return (D=C)
end if
if D=1 and C=1 **then** return (A=B)
end if
Pick a variable z
Do the next 4 lines in parallel
x = not(IsEqual($A_{z=0}, D_{z=0}, B_{z=0}, C_{z=0}$))
y = not(IsEqual($\frac{\partial A}{\partial z}$, $\frac{\partial D}{\partial z}$, $\frac{\partial B}{\partial z}$, $\frac{\partial C}{\partial z}$))
z = IsEqual($\frac{\partial A}{\partial z}, B_{z=0}, A_{z=0}, \frac{\partial B}{\partial z}$)
w = IsEqual($\frac{\partial A}{\partial z}, C_{z=0}, A_{z=0}, \frac{\partial C}{\partial z}$)
Wait for all threads to finish
if not(x) **then** return FALSE
end if
if not(y) **then** return FALSE
end if
if z **then** return TRUE
end if
if w **then** return TRUE
else return FALSE
end if

4.1 Logic Circuit Synthesis Benchmarks

We used ITC'99 [2], Iscas'85 [9], and n-bit ripple carry adder [12] benchmarks. RTL designs of the digital logic circuits were converted from Verilog to the full disjunctive normal form to obtain the corresponding Boolean polynomial. The sequential and parallel algorithms were evaluated on the obtained Boolean polynomials. Table 1 shows the execution time of sequential and parallel algorithms executed on Xeon processor running at 2.8 GHz with 4 threads averaged over 5 runs. One can observe a considerable performance speedup of the parallel algorithm over the sequential one.

Table 1. Results on Xeon processor at 2.8 GHz using 4 threads

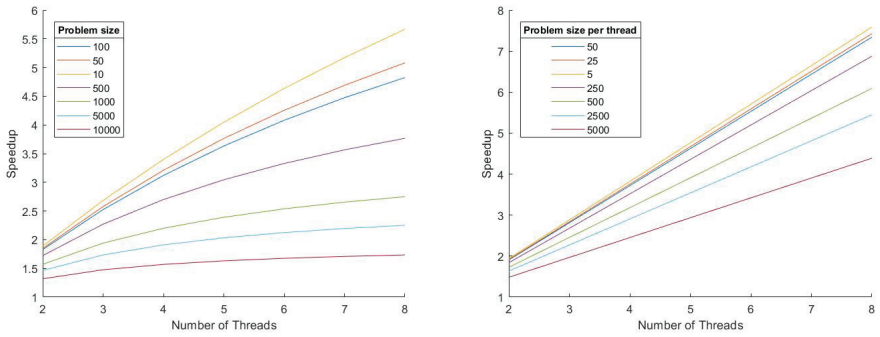
Benchmark	**Sequential**	**Multi-Threaded**	**Speedup**
ITC'99	4324(s)	1441(s)	3.01
Iscas'85	7181(s)	2633(s)	2.73
EPFL Adder	1381(s)	374(s)	3.69

4.2 Synthetic Polynomials

Synthetic polynomials of varying complexities were generated at random and sequential and parallel algorithms were evaluated on them. Table 2 shows execution times for the sequential and parallel algorithms executed on Xeon processor running at 2.8 GHz with 4 threads averaged over 5 runs. We observe that the execution time of both sequential and multithreaded algorithm increases drastically with the increase in the complexity of Boolean polynomials. We also observe that the speedup due to parallelization decreases with the increase in the complexity of Boolean polynomials.

Table 2. Execution time of factoring synthetic polynomials on Xeon processor at 2.8 GHz using 4 threads

Number of Monomials	**Sequential**	**Multi-Threaded**	**Speedup**
10	0.023(s)	0.0074(s)	3.12
50	16.29(s)	5.07(s)	3.21
100	103.5(s)	30.44(s)	3.4
500	483.6(s)	178.1(s)	2.7
1000	1165(s)	520.9(s)	2.2
5000	1430(s)	735.11(s)	1.91
10000	12614(s)	8034(s)	1.57



(a) (b)

Fig. 1. (a) Parallel speedup vs number of threads with fixed problem size
(b) Parallel speedup vs number of threads with fixed problem size per thread

4.3 Scaling Results

Figure 1a shows the speedup of the parallel decomposition algorithm over the sequential one wrt the number of threads. Here, the problem size is fixed to examine the strong scaling behaviour of the parallel decomposition algorithm. We observe that the parallel speedup is decreased as the size (complexity) of the problem increases. As the problem size increases, so does the call to the sequential bottleneck of the algorithm (simplification of Boolean polynomials), which causes the speedup to reduce.
Figure 1b shows the speedup of the parallel decomposition algorithm over the sequential algorithm wrt the number of threads. Here, the problem size per thread is fixed to examine the weak scaling behaviour of the parallel algorithm. The increase in the parallel speedup with the increase in the number of threads is less than the ideal linear speedup. This is due to the sequential bottlenecks in the decomposition algorithm (simplification of Boolean polynomials) and the communication bottleneck among multiple threads. Note that in these tests number of variables ranges from tens to two hundreds.

5 Implementation on Redefine

The REDEFINE architecture [1] comprises Compute Resources (CRs) connected through a Network-on-Chip (NoC) (see Figure 2a). REDEFINE is an application accelerator, which can be customized for a specific application domain through reconfiguration. Reconfiguration in REDEFINE can be performed primarily at two levels, viz. the level of aggregation of CRs to serve as processing cores for coarse grain multi-input, multi-output macro operations, and at the level of Custom Function Units (CFU) presented at the Hardware Abstraction Layer (HAL) as Instruction Extensions. Unlike traditional architectures, Instructions

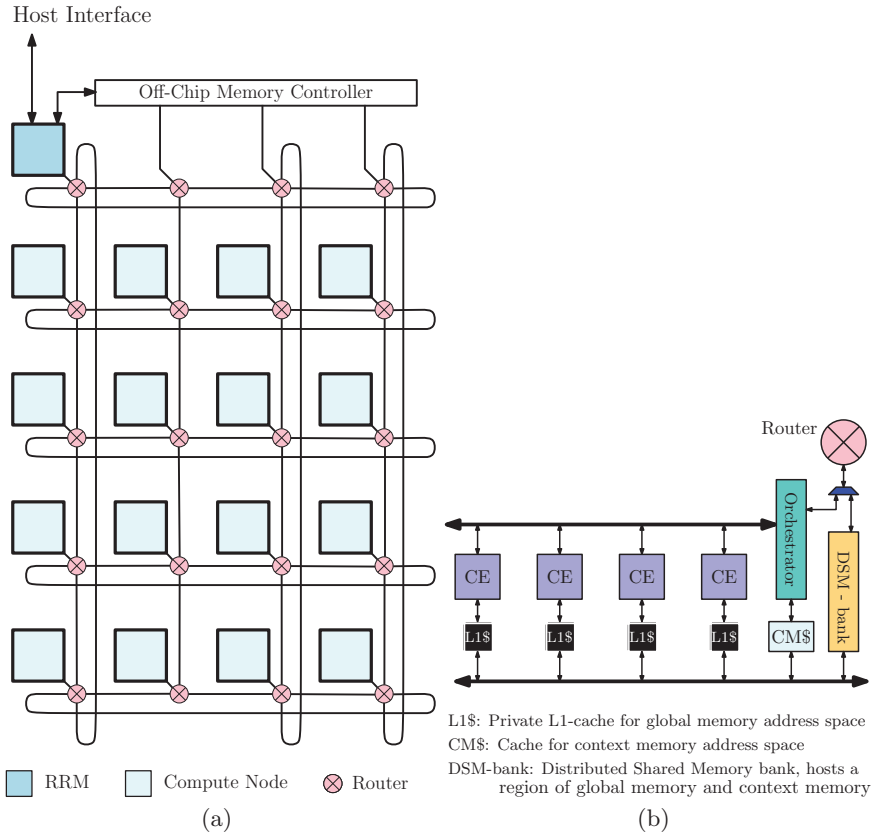


Fig. 2. (a) A 16 node REDEFINE comprising of a 4x4 toriodal mesh of routers and a redefine resource manager(RRM) for interfacing with the host
(b) Composition of a compute Node

Extensions in REDEFINE can be defined post-silicon. Post-silicon definition of Instruction Extensions in REDEFINE is a unique feature of REDEFINE that sets it aside from other commercial multicores by allowing customization of REDEFINE for different application domains.

REDEFINE execution model is inspired by the macro-dataflow model. In this model, an application is described as a hierarchical dataflow graph, as shown in Figure 3, in which the vertices are called hyperOps, and the edges represent explicit data transfer or execution order requirements among hyperOps. A hyperOp is a multiple-input and multiple-output (MIMO) macro operation. A hyperOp is ready for execution as soon as all its operands are available and all its execution order or synchronization dependencies are satisfied. Apart from the arithmetic, control, and memory load and store instructions, the REDEFINE execution model includes primitives for explicit data transfers and synchronization among hyperOps and primitives for adding new nodes (hyperOps)

and edges to the application graph during execution. Thus, the execution model supports dynamic (data-dependent) parallelism. The execution model follows non-preemptive scheduling of hyperOps; therefore cyclic dependencies are forbidden among hyperOps. The runtime unit named Orchestrator schedules ready hyperOps onto CRs. A CR comprises four Compute Elements (CEs). Each CE executes a hyperOp (see Figure 2b). All communications among hyperOps are unidirectional i.e., only producer hyperOp initiates and completes a communication. Thus with sufficient parallelism, all communications can overlap with computations. Compared to other hybrid dataflow/control-flow execution models, REDEFINE execution model simplifies the resource management and the memory model required to support arbitrary parallelism.

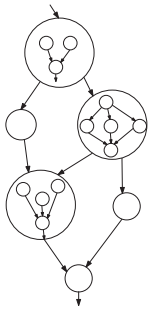


Fig. 3. Macro-dataflow execution model. An application described as a hierarchical dataflow graph, in which vertices represent hyperOps and edges represent explicit data transfer or execution order requirements between the connected hyperOps.

5.1 Decomposition Algorithm Using HyperOps

Algorithm 5 describes in pseudo-code the decomposition algorithm when written using "C with HyperOps". The code snippet, corresponding to Algorithm 5 is presented in the listing below. In the code snippet the terms `__CMAddr, __Sync, __kernel,__WriteCM, CMADDR` are REDEFINE specific annotations. The Lines 2-8 and 12-13 of Algorithm 5 are the same as Lines 5-10 and 1-3 of Algorithm 1, respectively. In Lines 15-17 of Algorithm 5, for each variable y in the variable set of F (excluding x) we spawn HyperOps in parallel to calculate whether y belongs to Σ_{same} or Σ_{other}. In Lines 1-10, we define the HyperOp. It takes as input Boolean polynomials A, B and a variable y and adds y to Σ_{same} or Σ_{other}. In Lines 18-20, we wait for the all the HyperOps to finish and output F_{same} and F_{other}.

The proposed decomposition algorithm using HyperOps was evaluated using REDEFINE emulator executed on Intel Xeon processor. Table 3 shows the execution time of the decomposition algorithm executed on Redefine emulator

Algorithm 5 Decomposition Algorithm using HyperOps

Input Boolean polynomial to be factored F
Output F_{same} and F_{other} which are the factors of the input polynomial F
Global variables $\Sigma_{same}, \Sigma_{other}$

1: **Begin HyperOp**
2: Inputs: A, B, variable y
3: Calculate $C = \frac{\partial A}{\partial y}$, $D = \frac{\partial B}{\partial y}$
4: **if** IsEqual(A,D,B,C) **then**
5: $\Sigma_{other} = \Sigma_{other} \cup \{y\}$
6: **else**
7: $\Sigma_{same} = \Sigma_{same} \cup \{y\}$
8: **end if**
9: Call Sync HyperOp
10: **End HyperOp**
11: Take an arbitrary variable x occurring in F
12: Let $A = \frac{\partial F}{\partial z}, B = F_{z=0}$
13: Let $\Sigma_{same} = x, \Sigma_{other} = \emptyset, F_{same} = 0, F_{other} = 0$
14: **for each** $y \in var(F) \setminus \{x\}$ **do in parallel**
15: Spawn HyperOp with inputs A, B, y
16: **end for**
17: Wait for the Sync HyperOp to return
18: If $\Sigma_{other} = \emptyset$ then F is non-factorable; stop
19: Return polynomials F_{same} and F_{other} obtained as projections onto Σ_{same} and Σ_{other}, respectively.

Table 3. Parallel factoring of synthetic boolean polynomials using REDEFINE emulation running on Intel Xeon processor at 2.8 GHz

Number of Monomials	**Sequential (cpu cycles)**	**Multi-Threaded (cpu cycles)**	**Redefine (cpu cycles)**
30	17192×10^3	7896×10^3	6837×10^3
50	45612×10^3	14196×10^3	12320×10^3

on synthetic Boolean polynomials. The Redefine implementation has the lowest CPU cycles.

6 Conclusions and Future Work

In this paper, we have reviewed the factorization problem for Boolean polynomials. Factorization provides the basis for decomposition of Boolean functions in different representations and decomposition of data tables. Hence, it is important to develop efficient factorization procedures. We have considered the approach presented in [4] for factoring Boolean polynomials and presented a MIMD implementation thereof, which exploits task and data level parallelism to achieve better performance. Evaluation of the sequential and parallel algorithms on logic circuit synthesis benchmarks and synthetic Boolean polynomials

showed a considerable speedup obtained by parallelization. The implementation of the parallel algorithm on a REDEFINE emulator outlined the performance benefits under execution on a massively parallel many core architecture. REDEFINE execution model is based on data flow principles and hence, the need for explicit barrier synchronization is obviated. This results in better performance of MIMD applications (Ex:Boolean factorization) on the REDEFINE architecture. In the future work we are going to benchmark the proposed parallel algorithm on REDEFINE hardware and analyze the results. We also plan to use REDEFINE for an efficient hardware implementation of Boolean functions given as Boolean polynomials and DNFs in order to efficiently implement decomposition algorithms for these representations. Finally, we are going to use these implementations for non-disjoint decomposition of DNFs [11] and data tables [3], which is based on massive computation of disjoint decompositions as a subtask. Listing 1.1 describes the decomposition program written using C with HyperOps.

```
__hyperOp__ void decompose(__CMAddr selfId, __Op32 a, __Op32 b,__Op32 p_s,__Op32 p_o,
      __Op32 m,__Op32 n, __Op32 i, __Op32 consumerFrId){

   int *A   = a.ptr;
   int *B   = b.ptr;
   int *partition_same = p_s.ptr;
   int *partition_other = p_o.ptr;
   int I = i.i32;
   int n = n.i32;
   int m = m.i32;
   int I=0,J = 0;
   int *C, *D;

   __CMAddr confrId = consumerFrId.cmAddr;
  for (I = 0; I<n; I++){
       *(partition_same+J) = 0;
       *(partition_other+J) = 0;
    }

  for (I = 0; I<m; I++){
    for(J=0;J<n;j++){
       *(C+I*columns+J) = 0;
       *(D+I*columns+J) = 0;
     }
   }
   derivative(A,B,C,i);
   derviative(A,B,D,i);
   if(IsEqual(A,D,B,C)){
    *(partition_other+i) =1;
   }
   else{
        *(partition_same+i) =1;
   }
   __Sync( confrId, -1);
}


__kernel int decompose_start (int *A, int *B,int *partition_same, int *
     partition_other, int N){

    int i = 0, j = 0;
    static int counter = 0;
    __CMAddr decomposeFr;
    __CMAddr syncFr = __CreateInst(&smd_Sync);
    __WriteCM( CMADDR(syncFr, 15), N-1);

  for (i = 1; i<N; i++){
    decomposeFr = __CreateInst(&smd_decompose);
    __WriteCM( CMADDR(decomposeFr, 0), (void*)(A));
    __WriteCM( CMADDR(decomposeFr, 1), (void*)(B));
    __WriteCM( CMADDR(decomposeFr, 2), (void*)(partition_same));
    __WriteCM( CMADDR(decomposeFr, 3), (void*)(partition_other));
    __WriteCM( CMADDR(decomposeFr, 4), M);
    __WriteCM( CMADDR(decomposeFr, 5), N);
    __WriteCM( CMADDR(decomposeFr, 6), i);
    __WriteCM( CMADDR(decomposeFr, 7), CMADDR(syncFr,15));
    }
    return 0;
}
```

Listing 1.1. Snippet of Decomposition algorithm using Hyperops

References

1. Redefine - reconfigurable silicon core description. http://morphing.in/redefine, accessed: 2018-12-07
2. Corno, F., Reorda, M., Squillero, G.: Rt-level itc'99 benchmarks and first atpg results. Design Test of Computers, IEEE **17**(3), 44–53 (Jul 2000). https://doi.org/10.1109/54.867894
3. Emelyanov, P.: On two kinds of dataset decomposition. In: Proceedings of the 18^{th} International Conference on Computational Science (ICCS 2018), Part II. Lecture Notes in Computer Science, vol. 10861, pp. 171–183. Springer (2018)
4. Emelyanov, P., Ponomaryov, D.: Algorithmic issues of AND-decomposition of boolean formulas. Programming and Computer Software (2015)
5. Emelyanov, P., Ponomaryov, D.: On tractability of disjoint AND–decomposition of boolean formulas. In: Proceedings of the PSI 2014: 9^{th} Ershov Informatics Conference. Lecture Notes in Computer Science, vol. 8974, pp. 92–101. Springer (2015)
6. Emelyanov, P., Ponomaryov, D.: Cartesian decomposition in data analysis. In: Siberian Symposium on Data Science and Engineering (SSDSE) (2017)
7. Emelyanov, P., Ponomaryov, D.: On a polytime factorization algorithm for multilinear polynomials over f2. In: Gerdt, V.P., Koepf, W., Seiler, W.M., Vorozhtsov, E.V. (eds.) Computer Algebra in Scientific Computing - 20th International Workshop, CASC 2018, Lille, France, September 17-21, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11077, pp. 164–176. Springer (2018). https://doi.org/10.1007/978-3-319-99639-4, https://doi.org/10.1007/978-3-319-99639-4_11
8. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press, New York, NY, USA, Third edn. (2013)
9. Hansen, M.C., Yalcin, H., Hayes, J.P.: Unveiling the iscas-85 benchmarks: A case study in reverse engineering. IEEE Des. Test **16**(3), 72–80 (Jul 1999). https://doi.org/10.1109/54.785838, https://doi.org/10.1109/54.785838
10. Muller, D.E.: Application of Boolean algebra to switching circuit design and to error detection. IRE Transactions on Electronic Computers **EC-3**, 6–12 (1954)
11. Ponomaryov, D.: A polynomial time delta-decomposition algorithm for positive dnfs. In: Proceedings of the 14^{th} International Computer Science Symposium in Russia (CSR). Lecture Notes in Computer Science, vol. 11532. Springer (2019)
12. Schmidt, J., Fišer, P.: A prudent approach to benchmark collection
13. Shpilka, A., Volkovich, I.: On the relation between polynomial identity testing and finding variable disjoint factors. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) Automata, Languages and Programming (2010)
14. Zhegalkin, I.: Arithmetization of symbolic logics. Sbornik Mathematics **35**(1), 311–377 (1928), in Russian.

The measure of regular relations recognition applied to the supervised classification task

Yuri Mikheev[1][0000-0002-7641-6243]

[1] Ph.D., Saint – Petersburg State University

Abstract.

The probability measure of regularities recognition in an information stream is introduced in the paper. The measure allows creating machine-learning models without a supervisor. The experiment described in the paper proves that the measure allows recognizing regularities and finding out regular relations between values of variables.

Machine learning models find out regular relations in datasets that allow reconstructing unknown values of the classification variable. The classification algorithm, based on the probability measure of regularities recognition, is described in the paper. The measure of connection with entropy is demonstrated, and mutual information is used to optimize algorithm performance. The accuracy of the algorithm fits the accuracy of well-known supervised machine learning algorithms and even exceeds it.

Keywords: Cognition, Machine-Learning, Classification, Supervised Classification Algorithm, Prediction, Regularity, ZClassifier, Association rules.

1 Human and Machine Cognition

Before computers and computer science were developed, we could research cognition activity only by studying human beings. From the middle of the 20th-century, scientists have been able to make computer models of cognition and then test them [1].

Now, because of the growth in the power of computers and the availability of datasets, it has become easier to make and test various cognition activity models. And it appears that many machine-learning algorithms are being widely used [1,2,4,7,8,14,15,18,19].

Cognition is an essential factor influencing survival. Prediction is a result of cognition activity, and prediction accuracy is the measure of the quality of cognition. So, if the model cognition activity makes a good prediction, it could be considered as realistic and functionally relevant to the original cognition subject.

In this paper, cognition is considered as a process of regularities recognition in the input information stream. Cognition system finds it, memorizes it and then uses it for prediction.

Entropy as a measure of certainty is widely used in supervised machine learning [10,15]. Here, we use entropy for unsupervised machine learning to select regularities needed for prediction.

2 Probability measure of regular relations recognition

Let us use the following operational definition - regularity is an often-observed, repeated relation between symbols in an information stream.

We consider relations in the same way as it is/they are considered in the set theory; for example, relations of order, similarity, identity, etc. In the paper, only similarity relation is used, but the same approach could be easily applied to any other relation. The considered measure could be applied to any relation or set of relations between symbols in the information stream.

There are simple regularities between two symbols and more complex relations. The more the number of symbols in a relation, the more complex the relation is.

We should consider that regularity exists if the probability of observation relation between symbols is more than random. We title the measure of difference from the random probability as "The measure of regularities recognition".

Symbols in the information stream have the empirical probability of appearing p_s,. $s \in S$ is a subset of all possible input symbols or input dictionary. $s_z \in S$ – the subset of the symbols included in regularity. Defining p_{s_z} as the probability of the regularity between symbols in s_z,

The measure of regularities recognition is

$$\frac{p_{s_z}}{p'_{s_z}}$$

And in the logarithm form

$$Z = \log\left(\frac{p_{s_z}}{p'_{s_z}}\right)$$

Where $p_{s_z} = \frac{f(s_z)}{m}$, $f(s_z)$ – frequency of the relation appearance, $m -$ amount of observations in a data stream, p'_{s_z} - probability of appearance $s_z \in S$ as if they are independent. From the theorem of probabilities multiplication, it follows that the probability of the independent events is a multiplication of probabilities of each of them:

$$p'_{s_z} = \prod_{s \in s_z} p_s$$

Then

$$Z = \log\left(\frac{p_{s_Z}}{\prod_{s \in s_Z} p_s}\right) \quad (1)$$

The measure shows how large is the difference between the regular and the random appearances of the symbol s_z.

The measure is closely related to function lift in association rule algorithms. In association rule works, lift is used as one of the a priori criteria for selecting rules[12-14], and we are using it as a measure of sufficiency of information for prediction.

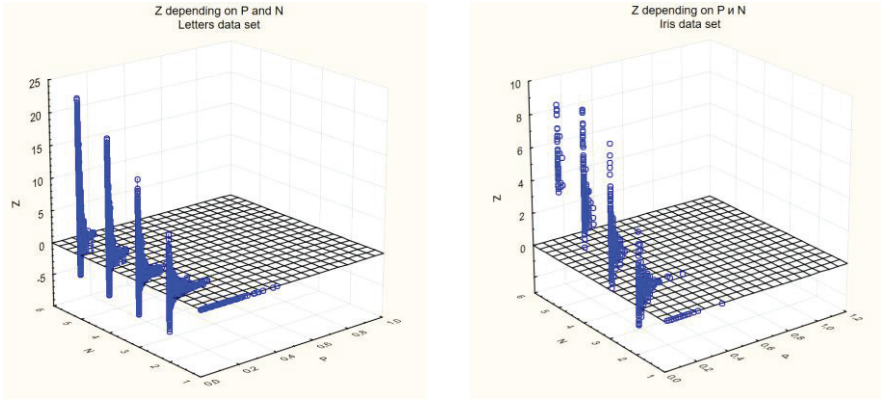


Fig. 1. Z depending on p_{s_z} and $N = |s_z|$

Expression (1) in the form of conditional probabilities:

$$Z = \log\left(\frac{p_{s_Z}}{\prod_{s \in s_Z} p_s} = \frac{p\left(s_1 \middle| C_{-1} = \{s_2, s_3, \dots, s_n\}\right) * p(C_{-1})}{\prod_{s \in s_Z} p_s}\right) = \quad (1.1)$$

$$= \log\left(\frac{p(s_1|C_{-1}) * p(s_2|C_{-2}) * \dots * p(s_{n-1}|s_n) * p(s_n)}{\prod_{s \in s_Z} p_s}\right) =$$

$$= \log\left(\frac{p(s_1|C_{-1}) * p(s_2|C_{-2}) * \dots * p(s_{n-1}|s_n)}{\prod_{s \in s_Z \cap \overline{s_n}} p_s}\right)$$

Where $C_{-1} = \{s_2, s_3, \dots, s_n\}$ is a subset s_z excluding $s_1$that is $s \in s_z \cap \overline{s_1}$. $C_{-2} = \{s_3, s_4, \dots, s_n\}$, – is a subset s_z excluding s_2 that is $s \in s_z \cap \overline{s_1} \cap \overline{s_2}$, and so on.

So, Z is dependent on joint conditional probabilities $s \in s_z$. Therefore, Z could help select a set of symbols that have the strongest influence on each other. (1.1) is a very important property for algorithm optimization and selecting the most important symbols for prediction.

If Z < 0 then:

$$\frac{p_{s_z}}{\prod_{s_z} p_s} < 1$$

Regularity is not found because of probability regularity appearance being less than the random appearance of symbols in relation s_z.

If Z ≥ 0 then:

$$\frac{p_{s_z}}{\prod_{s_z} p_s} \geq 1 \quad (2)$$

Regularity is recognized in the input data stream.
Thus:

$$Z \in (-\infty, 0) \text{ – Random relation of symbols}$$
$$Z \in [0, \infty) \text{ – Regular relation } s_z$$

2.1 Related works

Association rules mining is most related to the paper [22]. Many works were published in the field of association rules analysis for the classification problem, for example [9,18,21,22]. The major problem of those works is reducing the number of rules in model and selecting the most useful rules for classification. We deal with a similar problem in this paper. Unlike the mentioned works, entropy and the amount of mutual information for rule selection are used in this paper.

In association rules approach, terms unusual for probabilistic theory are widely used. The most popular of them are support(X) which is the same as the probability of the X in dataset and confidence(X => Y) which is a conditional probability. In this paper, classical probability theory terms are used in order to link information theory and informational entropy.

Entropy approach in the form of information gain is widely used in decision tree algorithms and regression [5, 8 ,10, 15]. All decision tree algorithms are supervised machine learning kinds of algorithms. The way for using entropy approach to unsupervised learning is presented here. In the paper, the amount of mutual information is used for calculating values of the classification variable.

Mutual information is a basis of Bayesian networks. Supervised classification algorithms use conditional information in oriented acyclic graphs. Instead of that, we are directly using for unsupervised learning mutual information from relations of variables values.

2.2 Classification task

In a classification task, we have a dataset D containing m records with observations of variables $x \in X$. We can calculate $c \in \mathrm{C}$ that are all observed combinations of the values of the variables in the data set.

Let us be sure that, the mathematical expectation of Z of $c \in C$ is the mutual information between variables in the dataset.

Mutual information between two systems [6, 16]:

$$I_{X \leftrightarrow Y} = E\left[\log \frac{P(X,Y)}{P(X)P(Y)}\right]$$

In this case, systems are subsets of the variables in the dataset. If we put in this expression the variables from X, we get:

$$I_{X \leftrightarrow X} = E\left[\log \frac{P(x_1,x_2,\ldots,x_n)}{\prod_{x \in X} P(x)}\right] = E[\mathrm{Z}] \quad (3)$$

From attributes of the mutual information [6, 16], we get

$$I_{X \leftrightarrow X} = E[Z] = \sum_{x \in X} H(x) - H(C),$$

Where C is a set of relations on the set of values of variables $x \in X$, $H(x)$ – entropy variable x, $H(C)$ – entropy relations between values of x.

Let's express it by conditional entropy. We know

$$H(X,Y) = H(X) + H(Y|X)$$

thus

$$H(C) = \sum_{x \in X} H(x) + \sum_{x \in X} \sum_{r \in X} H(x\,|r),$$

$$I_{X \leftrightarrow X} = E[Z] = \sum_{x \in X} H(x) - H(C) = \sum_{x \in X} H(x) - \sum_{x \in X} H(x) -$$

$$- \sum_{x \in X} \sum_{r \in X} H(x\,|r)$$

$$E[Z] = - \sum_{x \in X} \sum_{r \in X} H(x\,|r) = I_{X \leftrightarrow X} \quad (3.1)$$

So, if we calculate E[Z] of variables or any set of symbols included in C, we will have an amount of information concentrated either in the set of the variables or the set of symbols. Amount of mutual information shows how strong are interdependent values of the variables and in what degree they could be used for reinstating one of them, for example, classification variable.

The following hypothesis should be tested:

1. Expression (1) could be used for recovering relations between the values of the variables in data sets and reinstating unknown values in the investigated variable.
2. By using expression (2), we can create algorithms that recognize and use regularities for prediction.

If we succeed in it, we empirically prove the truth of (1) and the usefulness of (2).

Let's be sure of it and make the algorithm on the basis of (1) and (2) and title the algorithm "ZClassifier".

3 Experimental testing of The measure of regularities recognition

In order to test the measure of regularities recognition, we'll solve the problem of supervised classification and use published datasets. If ZClassifier does accurate classifications, then we prove the correctness of the measure of regularities recognition.

Using (1), we calculate how strongly interconnected are the values of variables in the input dataset.

Let data in the dataset be structured as a record $r_k \in D$. Every record contains values of the variables x_i, $i \in 1, \dots, n$, n – a number of variables in the dataset.

Let S be the set of values of the variables $x_i \in X$, s_{x_i} – set of values of the variables x_i. S is a dictionary of input dataset and ${s^l}_{x_i}$ is a symbol from the dictionary. Every record in the input dataset contains a set of relations between symbols of the records c_k which is the powerset $(\{s^l_{x_i} | \, s^l_{x_i} \in r_k\})$. $\forall c_k \in C$, C is the set of all relations acquired in the dataset.

For each combination, we can calculate probability (c_k) , and expression (1) becomes:

$$Z(\mathrm{c}_k) = \log\left(\frac{p(c_k)}{\prod_{s_{x_i}} p_{s^l_{x_i}}}\right) \tag{4}$$

$p_{s^l_{x_i}}$ is a probability appearance of ${s^l}_{x_i}$ in the dataset.

Z (4) shows how strong the variable values' interdependencies are.

The direct algorithm of making model is:

Input: Training dataset

For each record from the training dataset, do
    For each relation c in the record
        Increment appearance frequency *f(c_k)*
        Save c_k and *f(c_k)* in the model

The direct algorithm collects all relations existing in the dataset. However, not all of them are necessary to make an accurate prediction.

For an accurate prediction, a model needs only relations correlated with the classification variable. For the selection, correlated relations' partial joint information of relation could be used $I_{c_n} = p_{c_n} Z(c_n)$. When model collects enough information to predict X, that is, to explain entropy of X, the algorithm should stop. The following algorithm is based on this idea.

The back index algorithm of making model is:
Input: Training dataset D, I_t - mutual information threshold

Create back index of the training dataset
For all variables $x \in D$ calculate $I = -\sum_{x \in D} p(x)\log(p(x))$
For all relations, c of the values of variables $x, y \in D$ calculate mutual information $I_c = p_c Z(c)$ and save c in sorted list TC (relation with highest mutual information I_c is on the top of the list)
For all $c_i, c_j \in TC$ do
Make $c_n = c_i \cup c_j$ and calculate $I_{c_n} = p_{c_n} Z(c_n)$ by back index
If ($I_{c_n} > I_t$) save c_n in TC and increment $I_{tc} = I_{tc} + I_{c_n}$
Repeat till $I_{tc} < I \; or \; TC \neq \emptyset$

Value of classification variable recovering algorithm is:
Input: Testing Dataset, $s_{\acute{x}}$ values (symbols) of $\acute{x}$ are the investigated variables

For each record r from test dataset
Select all relations among ${s^l}_{x_i} \in r$ and save it in C_0
For all $c_k \in \mathrm{C}_0$ and each ${s^l}_{\acute{x}} \in S$ create relation $c_k \cup {s^l}_{\acute{x}}$,
and calculate $Z(c_k \cup {s^l}_{\acute{x}})$,
If $Z(c_k \cup {s^l}_{\acute{x}}) > 0$ then saves $c_k \cup {s^l}_{\acute{x}}$ in $\mathrm{C}_{\acute{x}}$
$\acute{x} = argmax(Pf({s^l}_{\acute{x}}))$, where $Pf(s_{\acute{x}})$ one of following functions:

a. $Pf({s^l}_{\acute{x}}) = \max_{c_k \in \mathrm{C}_0} (Z(c_k \cup {s^l}_{\acute{x}}))^a \, p'(c_k \cup {s^l}_{\acute{x}})^b)$
b. $Pf({s^l}_{\acute{x}}) = \sum_{c_k \in \mathrm{C}_0} (Z(c_k \cup {s^l}_{\acute{x}}))^a \, p'(c_k \cup {s^l}_{\acute{x}})^b$
c. $Pf({s^l}_{\acute{x}}) = \frac{1}{|\mathrm{C}_0|} \sum_{c_k \in \mathrm{C}_0} (Z(c_k \cup {s^l}_{\acute{x}}))^a \, p'(c_k \cup {s^l}_{\acute{x}})^b$,

a and b are experimentally fit constants, $p' = \frac{p(c_k \cup {s^l}_{\acute{x}})}{p(c_k \in \mathrm{C}_0)}$ conditional probability of $c_k \cup {s^l}_{\acute{x}}$

We are choosing a prediction function through experiments in order to maximize the prediction accuracy of the $\acute{x}$.

If we deal with numeric values of x_i (not categorical values), we should define intervals in order to decrease the amount of ${s^l}_{x_i}$. In that case, accuracy is dependent on the interval borders.

3.1 Direct Classifier algorithm characteristics and optimization

3.1.1 Working time

The algorithm is sensitive to the number of variables in the dataset and true complexity of the data. Algorithm complexity is $\mathrm{O}\,(2^n * m)$, n – the number of variables, m - the number of records. Time and memory usage exponentially grow with the growth of the number of variables.

Restricting the maximum number of symbols in relations sufficiently decreases the working time. In most cases, 5-7 symbols max in relation are enough for excellent accuracy. Complexity in those cases is O $(\sum_i^n C_i^7 * m)$ significantly less O $(2^n * m)$.

Another way to optimize time performance is by reducing the number of variables in the dataset. In practice, to make a prediction, we only need variables that correlate with the classification variable. We can estimate how strongly values of variables influence each other by using (1.1). Variable value influence is input in the amount of mutual information of relations containing the variable value.

Variable value influence estimation algorithm:
Input: C is the set of all relations acquired in the dataset

for all $c \in C$
  for all s from c
    $influence(s) = p_c Z(c) - p_{c \cap \bar{s}} Z(c \cap \bar{s})$

The influence shows how important it is for predicting the variable value and deciding if it is it worth using the variable in the model or not. Variables could be selected by influence.

3.1.2 Model size

The result of the training stage is a huge amount of relations. To reduce model size, we can use (3.1) and select only informative relations at the pruning stage.

Pruning algorithm:
Input: C is the set of all relations acquired in the dataset
  $minInformation$ – minimal information of c

for all $c \in C$ (C is the set of all relations acquired in the dataset),
  if $p_c Z(c) < minInformation$, then remove c form C

Pruning allows reducing model size in several times without any losses in accuracy.

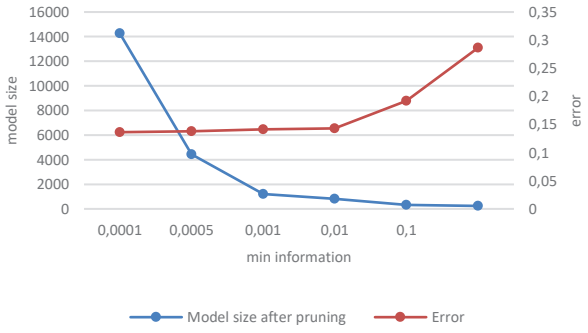


Fig. 2. Reducing model size and error growth after pruning (Adult dataset)

3.2 Back index ZClassifier algorithm complexity

Formally, algorithm complexity is O $(2^n * m * l^2)$, n – number of variables, m - number of records, l – number of relations in the sorted list. That is, the direct algorithm has lower complexity than the back index algorithm.

In practice, the back index ZClassifier algorithm works faster with datasets having a larger number of variables because the back

index algorithm deals with "good" relations that have a high amount of mutual information. A model improbable is needed to have 2^n relations for accurate prediction.

For example, Home Credit Default Risk competition dataset from Kaggle has 123 variable in the main data table. Direct ZClassifier can create a model that contains relations 3 symbols long max. Back index ZClassifier creates a model without restrictions on relation length in less than half an hour. Size comparison of the algorithms models is provided in Table 2 and the accuracy in Table 3.

.

4 The results of the experiments

ZClassifier was applied to several UCI datasets. Accuracy was estimated by kFold with most appropriate k to the dataset. Datasets are described in table 1.

Table 1. UCI dataset in the experiment

Dataset	Variables	Used variables	Records	kFold's k
Mushrooms	17	17	8 125	10
Adult	14	4	48 842	10
Census	40	4	299 291	10
Letters	17	15	20 000	10
Zoo	17	9	160	5 times k=2
Iris	4	4	149	5 times k=2
Segments	19	12	2310	10

k meanings in kFold were chosen in order to provide enough size of training data, so for small data sets, Zoo and Iris k was equal to 2, and we performed the experiment 5 times.

Table 2. Direct and Back Index algorithms complexity comparison

Dataset	Direct ZClassifier		Back Index ZClassifier	
	Relations in model	Relations after pruning	Relations in model	Relations after pruning
Mushrooms	53 168	5 108	3 737	2 634
Adult	12 920	1 622	2 314	699
Census	49 199	5 804	19 692	5 507
Letters	51 304 560	46 890 748	34 355	22 823
Zoo	2 028 923	-	743	-
Iris	608	608	253	219
Segments	461 255	-	1381	-

Comparing the ZClassifier results with those of other supervised classification algorithms [10] (table 2):

Table 3. The comparison of the result accuracy

Dataset	NB-Tree	Naïve Bayes	C4.5	Gradient boosting	CMAR [21]	Direct ZClassifier	Back index ZClassifier
Mushrooms	100	95,52	100	100		100 ±0	93,28 ±1,73
Adult	85,9	83,88	85,54	87,53		86,18 ±0,91	86,1 ±0,75
Census		76,8	95,2	95,4		94,87 ±0,16	94,87±0,17
Letter	85,66	64,07	88,03	93,4		91,42 ±0,40	56,09±1,35
Zoo	93,07	94,97	92,61		97,1	97,42 ±0,45	95,0±3,0
Iris	95,4	95,53	94,73	95.83	94	91,92 ±0,75	92,8±2,84
Segments	95,34	80,17	96,79	99,9		96,22 ±0,31	76,2±5,0

From the results, we can conclude that classification on the basis of (1) have a comparable or better accuracy [13].

Gradient boosting [17] shows better results, but the boosting process takes a lot of time for iterations. ZClassifier finds out good decision in one iteration and could be boosted also in a similar way as other classification algorithms. This task is planned to be undertaken in the future.

5 Conclusions

The measure of regularities recognition is introduced in the paper. ZClassifier is an algorithm created in order to demonstrate that the measure of regularities recognition is working, and the measure shows good results in the classification task.

The hypothesis that expression (1) could be used for recovering relations between values of the variables in data sets and reinstating unknown values investigated variable and the hypothesis that using (2), we can create algorithms that recognize and use regularities for prediction, are proven.

The experiment shows that condition (2) decreases the algorithm work time. And pruning based on most informative relations selection allows a sufficiently reduced model size.

The measure of regularities recognition could be applied for regularities recognition in data streams having various structures of data. For example, the author created the algorithm for recognition regularities in English and Russian texts. The algorithm can distinguish words and regular phrases from the texts.

An example of entropy approach to unsupervised machine learning was demonstrated in the paper. The measure of regularities recognition could also be used in other algorithms. In the future, on the basis of the measure of regularities recognition, we will create unsupervised machine-learning algorithms for various types of datasets.

References

1. Ben-Hur Asa [et al.] Support vector clustering [Journal]. - [s.l.] : Journal of Machine Learning Research, 2, 2001. - 2. - pp. 125-137.
2. Bertsekas Dimitri P. Dimitri P. Bertsekas. "Dynamic Programming and Optimal Control: Approximate Dynamic Programming, Vol.II [Book]. - [s.l.] : Athena Scientific, 2012.
3. Blakeslee Jeff Hawkins & Sandra On intelligence [Book]. - [s.l.] : Times Books, 2004.
4. Brown J. D. Principal components analysis and exploratory factor analysis – Definitions, differences and choices [Journal] // Shiken: JALT Testing & Evaluation SIG Newsletter. - 13 January 2009. - 1. - pp. 26-30.
5. Cai Xiongcai 18th European Conference on Machine Learning [Conference] // Level Learning Set: A Novel Classifier Based on Active Contour Models. - Warsaw, Poland : [s.n.], 2007.
6. E.S. Ventcel Teoria veroytnostey. Вентцель, Е. С. (1999). Теория вероятностей: учебник для вузов. [Book]. - Moscow : [s.n.], 1999.
7. Estivill-Castro Vladimir Why so many clustering algorithms – A Position Paper [Journal] // ACM SIGKDD Explorations Newsletter. - 2002. - 4(1). - pp. 65-75.
8. Freedman David A. Statistical Models: Theory and Practice [Book]. - [s.l.] : Cambridge University Press , 2005.
9. Hahsler Michael and Hornik, Kurt and Reutterer, Thomas Implications of probabilistic data modeling for rule mining [Journal] // Research Report Series / Department of Statistics and Mathematics, 14. - 2005.
10. Jiang Liangxiao and Li Chaoqun Scaling Up the Accuracy of Decision-Tree: A Naive-Bayes Combination [Journal]. - [s.l.] : JOURNAL OF COMPUTERS. - VOL. 6, NO. 7, JULY 2011.
11. Kliegr Tom´aˇs Quantitative CBA: Small and Comprehensible Association Rule Classification Models [Journal] // School of Electronic Engineering and Computer Science,Queen Mary University of London, United Kingdom, and Faculty of Informatics and Statistics, VSE, Czec. - 2017.
12. MacKay David J.C. Information Theory, Inference, and Learning Algorithms (First ed.). [Journal] // Cambridge University Press.. - 2003. - p. 34.
13. Rich Caruana Alexandru Niculescu-Mizil An Empirical Comparison of Supervised Learning Algorithms. [Journal]. - Ithaca, NY 14853 USA : Department of Computer Science, Cornell University, 2006.

14. Rish Irina. An Empirical Study of the Naïve Bayes Classifier. [Journal] // IJCAI Work Empir Methods Artif Intell. - 2001. - 3.

15. Rokach Lior and Maimon Oded Data Mining with Decision Trees. Theory and Applications [Book]. - Israel : Ben-Gurion University of the Negev, Tel-Aviv University, 2007.

16. Shannon C. and Weaver, W. The Mathematical Theory of Communication [Journal] // Bell System Technical Journal. - 1948. - 27. - pp. 379-423, 623-656.

17. Sigrist Fabio Gradient and Newton Boosting for Classification [Journal] // Lucerne University of Applied Sciences and Arts. - 2019.

18. Srikant Rakesh Agrawal and Ramakrishnan 20th International Conference on Very Large Data Bases, pages 487-499 [Conference] // Fast algorithms for mining association rules. . - Santiago, Chile : [s.n.], 1994. - pp. 487-499.

19. Stuart J. Russell Peter Norvig Artificial Intelligence: A Modern Approach [Book]. - [s.l.] : Prentice Hall, 2009.

20. Vapnik Vladimir The Nature of Statistical Learning Theory. [Book]. - [s.l.] : Springer Science & Business Media, 1999.

21. Wenmin Li Jiawei Han, Jian Pei CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules [Journal] // School of Computing Science, Simon Fraser Universit.

22. Xiaoxin Yin Jiawei Han CPAR: Classification based on Predictive Association Rules [Journal] // University of Illinois at Urbana-Champaign.

Hermes: A Reversible Language for Writing Encryption Algorithms (Work in Progress)

Torben Ægidius Mogensen

DIKU, University of Copenhagen
Universitetsparken 5, DK-2100 Copenhagen O, Denmark
torbenm@di.ku.dk

Abstract. We describe the programming language Hermes, which is designed for writing private-key encryption algorithms. Specifically, every program written in Hermes is reversible: It can run equally well forwards and backwards. This means that you only write the encryption algorithm and get the decryption algorithm for free. Hermes also ensures that all variables are cleared after use, so the memory will not contain data that can be used for side-channel attacks. Additionally, to prevent side-channel attacks that extract information from running times, control structures that may give data-dependent run times are avoided.

1 Introduction

Recent work [6] have investigated using the reversible language Janus [2, 11] for writing encryption algorithms. Janus is a structured imperative language where all statements are reversible. A requirement for reversibility is that no information is ever discarded: No variable is destructively overwritten in such a way that the original value is lost. Instead, it must be updated in a reversible manner or swapped with another variable. Since encryption is by nature reversible, it seems natural to write these in a reversible programming language. Additionally, reversible languages requires that all intermediate variables are cleared to 0 before they are discarded, which ensures that no information that could potentially be used for side-channel attacks is left in memory. But non-cleared variables is not the only side-channel attack used against encryption: If the time used to encrypt data can depend on the values of the data and the encryption key, attackers can gain (some) information about the data or the key simply by measuring the time used for encryption. Janus has control structures the timing of which depend on the values of variables, so it does not protect against timing-based attacks.

So we propose a language, Hermes, specifically designed for encryption. What Hermes has in common with Janus is reversible update statements, swap statements, and procedures that can be called both forwards and backwards. The main differences to Janus are that Hermes operates on integers of specified sizes, specifically 32 and 64-bit signed and unsigned integers, and there are no time-sensitive control structures. The syntax of Hermes resembles C, so programs will be readily readable by C programmers.

Figure 1 shows a Hermes implementation of TEA, a Tiny Encryption Algorithm [7] corresponding to the C code in Figure 2, which is taken from the Wikipedia page for TEA [10]. While the Hermes code resembles the C code, this does not mean that we can automatically convert C programs to Hermes: In general, C statements are not reversible, and their timing may depend on data. But if an encryption algorithm is designed to be reversible and immune to timing attacks, it will usually be simple to (manually) port to Hermes. But the purpose is not to port existing C implementations of cyphers to Hermes, but to allow cypher designers to develop their cyphers in a language that ensures both reversibility and immunity to timing attacks.

```
encrypt (u32 v[2], u32 k[4])
{
    u32 v0, v1, sum, k0, k1, k2, k3;
    const u32 delta = 0x9E3779B9;          /* key schedule constant */
    v0 <-> v[0]; v1 <-> v[1];                          /* set up */
    k0 += k[0]; k1 += k[1]; k2 += k[2]; k3 += k[3]; /* cache key */
    for (i=0; 32) {                          /* basic cycle start */
        sum += delta;
        v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
        i++;
    }                                        /* end cycle */
    k0 -= k[0]; k1 -= k[1]; k2 -= k[2]; k3 -= k[3]; /* clear locals */
    sum -= delta << 5;    /* alternatively, sum -= 0xC6EF3720 */
    v[0] <-> v0; v[1] <-> v1;                /* return coded values */
}
```

Fig. 1. TEA in Hermes

Note that while the C version needs a separate decryption procedure, this is not required in Hermes, as decryption is achieved by running the encryption procedure backwards. Apart from using the swap operator `<->` a lot, the Hermes code is very similar to the encryption part of the C code, except that the local variables are explicitly cleared. If they were not, an error would be reported when running the program. Note that constants do not need to be cleared.

2 Hermes

The Syntax of Hermes is shown in Figure 3.

A program consists of one or more procedures, where the procedure called `main` is the entry point of the program. Unlike in C, the `main` procedure has no arguments. Arguments to procedures are passed by reference and to avoid aliasing, no variable or array may be passed several times in the same call. For simplicity, we do not allow global variables, but future versions of Hermes may add these.

The values used in Hermes are variables or one-dimensional arrays the elements of which are of the types `i8, u8, i16, u16, i32, u32, i64` or `u64`,

```
void encrypt (uint32_t v[2], uint32_t k[4]) {
    uint32_t v0=v[0], v1=v[1], sum=0, i;  /* set up */
    uint32_t delta=0x9E3779B9;            /* key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];  /* cache key */
    for (i=0; i<32; i++) {                /* basic cycle start */
        sum += delta;
        v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
    }                                     /* end cycle */
    v[0]=v0; v[1]=v1;
}

void decrypt (uint32_t v[2], uint32_t k[4]) {
    uint32_t v0=v[0], v1=v[1], sum=0xC6EF3720, i; /* sum=32*delta */
    uint32_t delta=0x9E3779B9;            /* key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];  /* cache key */
    for (i=0; i<32; i++) {                /* basic cycle start */
        v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
        v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        sum -= delta;
    }                                     /* end cycle */
    v[0]=v0; v[1]=v1;
}
```

Fig. 2. TEA in C

representing signed and unsigned 32 or 64-bit two's complement numbers (corresponding to the C types `int8_t, uint8_t, int16_t, uint16_t, int32_t, uint32_t, int64_t` and `uint64_t`). Sizes of arrays must be specified when they are declared. All variables are local to procedures, and must be cleared to zero before the end of the procedure. If a local variable is not zero at the exit of the procedure, a run-time error is reported.

Constants are initialised with a value and can not be modified. Constants do not need to be zeroed before procedure exit.

The body of a procedure is a statement. This can be

- The empty statement (`;`),
- An update using one of the update operators `+=`, `-=`, `^=`, `<<=`, or `>>=`, where the last two operators are rotate-left and rotate-right. The root variable on the left-hand side is not allowed to occur elsewhere on the update statement (neither left-hand side nor right-hand side). For example, the statements `i += a[i];` and `a[a[i]] += 1;` are not allowed, but `a[i] += i;` is allowed, as `i` is not the root variable on the left-hand side. Additionally, if the variable on the left-hand side is a loop variable (see later), the right-hand side expression must be a constant expression. Rotates are done on the word size of the variable on the left-hand side. For example, if `x` is an 8-bit number, `x <<= 11` will rotate the 8-bit number 3 positions left. Rotates are not found as operators in C, but they are commonly used in cryptology, and they are reversible, so it is natural to include them in Hermes.
- Increment or decrement of a variable or array element. These are special cases of updates.

– A conditional update. In addition to the restrictions for unconditional updates, the root variable on the left-hand side may not occur in the condition, nor may it be a loop variable. To avoid value-dependent timing, the right-hand side is always evaluated and afterwards logically ANDed with the condition before using the result in an update. As such, the conditional update does not any power to the language, it just aids readability of otherwise somewhat cryptic code.
– A swap of two variables or array elements, using the swap operator `<->`. The root variables on either side may not occur in any index expression, nor may they be loop variables. For example, the statements `i <-> a[i];` and `a[a[i]] <-> j;` are not allowed, but `a[i] <-> a[j];` is allowed. A swap is implemented as three updates (using `^=`) to avoid introducing a temporary variable that might leak information.
– A conditional swap. In addition to the restrictions of the normal swap, neither root variable may occur in the condition. A conditional swap is implemented as three conditional updates, so it does not add power to the language, but the conditional swap is easier to read. Conditional swap is commonly used in elliptic-curve cryptography to avoid time-inconstant conditional statements.
– A block in curly braces, consisting of a number of declarations and a number of statements. Variables and array elements are initialised to 0 and they must be returned to 0 at the end of the block (otherwise a run-time error is issued).
– A for loop. This specifies the initial and final values for a counter variable and a body that will be executed until the counter variable reaches its final value. Updating the counter variable is, unlike in C, done in the body of the loop. The final value must be reached exactly, otherwise the loop continues. The counter variable is local to the for loop and need not be declared (it is always of type `i32`). The expressions for initial and final values for the loop counter must be constant expressions. Also, the loop counter may only be unconditionally updated with constant expressions, but it may be updated multiple times and with any update operator (`+=`, `-=`, `^=`, `<<=`, and `>>=`).
– An assertion. If the condition evaluates to false, a run-time error is issued. This is included for testing purposes.
– A procedure call. Arguments are passed by reference. No variable may be repeated in the argument list, so the statements `call f(i, i);`, `call f(i, a[i]);`, and `call f(a, a[i]);` are illegal. To avoid potential modification, loop variables can not be passed as arguments to procedures.
– An inverse procedure call. This executes the procedure in backwards order, so the sequence `call f(x); uncall f(x);` has no net effect.
– Print and scan statements. These use format strings like in C, except that the formats are `%u8`, `%i8`, `%u16`, `%i16`, etc. Before reading a variable or array element, this must have the value 0, otherwise, a run-time error is issued. After printing a variable or array element, this is set to 0. Loop variables and constants can not be printed or scanned.

Expressions are variables, array elements, constants, operators applied to expressions, or conditions. Constants are numbers in decimal or hexadecimal form,

using C notation. Binary operators are `+`, `-`, `*`, `/`, `%`, `&`, `|`, `==`, `!=`, `<`, `>`, `<=`, `>=`, `<<`, and `>>`. Unary operators are `-` and `~`. All operators have the same meaning as in C, and like in C, there is no separate Boolean type – 0 is treated as logical falsehood and all non-zero values as falsehood. Note that we do not include the logical operators `&&`, `||`, and `!`, as their timing may depend on the values of their arguments. Bitwise logical operators should be used instead.

3 More Examples

An implementation of the speck128 cipher [1, 9] in C and Hermes is shown in Figure 4.

Hermes has rotation built-in, so it does not need the `ROR` and `ROL` macros. But since Hermes does not support macros, `R` must be defined as a procedure. Since loop variables can not be modified, it is not allowed to pass them as parameters to procedures, so we use a variable `ii` to hold a copy of the loop variable `i`. The Hermes version does not use separate parameters for the original and encrypted text, since we want to use the encryption function in reverse for decryption. A complication compared to a normal C implementation is that the round keys `a` and `b` must be restored (in the second for-loop) so they can be reset to 0 before the procedure exits. Note the use of `uncall` to do `R` in reverse.

While the Hermes version is slightly larger than the C version, a C program would have to define separate functions for encryption and decryption.

To illustrate the use of conditional updates, Figure 5 shows a simple (and probably weak) shift-register-based cipher. Note that, since the condition in a conditional update may not involve the updated variable, the value of the condition is computed in a variable `c` before the update. To ensure reversibility of the procedure, `c` is returned (uncomputed) to 0 afterwards. We restrict `K[0]` to be even to make this uncomputation possible.

Figure 6 includes C and Hermes versions of the central part of the RC5 cipher [4,8], i.e., not including the key expansion part. Again, Hermes doesn't need the `ROL` macro, and we use a single parameter for the original and encrypted values (`pt, ct`), but we must pass the expanded keys (`S[]`) as a parameter since we don't have global variables. As usual, Hermes doesn't need a separate decryption function. The updates to `A` and `B` must in Hermes be done as sequences of reversible updates, which is slightly more verbose, but also makes it clearer that the transformations are, in fact, reversible.

4 Compiling Hermes to C

We have implemented a prototype of Hermes by writing a compiler from Hermes to C. Each Hermes procedure is compiled to two C functions: One for running forwards and one for running backwards. The backwards version of a procedure is compiled by first doing a source-level inversion of the Hermes procedure and then compiling the inverted procedure to C. A command-line option allows the

$Program \rightarrow Procedure^{+}$

$Procedure \rightarrow$ **id** `(` $Decls2^{?}$ `)` $Stat$

$Stat \rightarrow$ `;`
| $Lval$ **update** Exp `;`
| $Lval$ `++;`
| $Lval$ `--;`
| `if (` Exp `)` $Lval$ **update** Exp `;`
| $Lval$ `<->`$Lval$
| `if (` Exp `)` $Lval$ `<->`$Lval$
| `for (` **id** `=`Exp `;` Exp `)` $Stat$
| `assert (` Exp `);`
| `call` **id** `(` $Lvals$`)`
| `uncall` **id** `(` $Lvals$`)`
| `printf (` **stringConst** `,` $Lvals$ `);`
| `scanf (` **stringConst** `,` $Lvals$ `);`
| `{` $Decls1$ $Stat^{*}$`}`

$Exp \rightarrow Lval$
| **numConst**
| Exp **binOp** Exp
| **unOp** Exp
| `(` Exp `)`

$Lval \rightarrow$ **id**
| **id** `[` Exp `]`

$Lvals \rightarrow Lval$
| $Lval$ `,` $Lvals$

$VarSpec \rightarrow$ **id**
| **id** `[` **numConst** `]`

$VarSpecs \rightarrow VarSpec$
| $VarSpec$ `,` $VarSpecs$

$Decls1 \rightarrow$
| **type** $VarSpecs$ `;` $Decls1$
| `const` **type id** `=` **numConst** `;` $Decls1$

$Decls2 \rightarrow$ **type** $VarSpec$
| **type** $VarSpec$ `,` $Decls2$

Fig. 3. Syntax of Hermes

```c
#include <stdint.h>

#define ROR(x,r) ((x >> r) | (x << (64 - r)))
#define ROL(x,r) ((x << r) | (x >> (64 - r)))
#define R(x,y,k) (x = ROR(x,8), x += y, x ^= k, y = ROL(y,3), y ^= x)
#define ROUNDS 32

void encrypt(uint64_t ct[2],
             uint64_t const pt[2],
             uint64_t const K[2])
{
   uint64_t y = pt[0], x = pt[1], b = K[0], a = K[1];

   R(x, y, b);
   for (int i = 0; i < ROUNDS - 1; i++) {
      R(a, b, i);
      R(x, y, b);
   }

   ct[0] = y;
   ct[1] = x;
}
```

```
R(u64 x, u64 y, u64 k)
{ x >>= 8; x += y; x ^= k; y <<= 3; y ^= x; }

speck128(u64 ct[2], u64 K[2])
{
   u64 y, x, b, a, ii;
   y <-> ct[0]; x <-> ct[1]; b += K[0]; a += K[1];

   call R(x, y, b);
   for (i=0; 32) {
     call R(a, b, ii);
     call R(x, y, b);
     ii++; i++;
   }
   for (i=32; 0) {  /* restore a and b */
     i--; ii--;
     uncall R(a, b, ii);
   }
   y <-> ct[0]; x <-> ct[1]; b -= K[0]; a -= K[1];
}
```

Fig. 4. Spec128 in C (top) and Hermes (bottom)

```
shift(u64 v, u64 K[2])
{
   u64 a, b, c;
   a += K[0]; b += K[1];   /* K[0] must be even */
   for (i=0; 13) {
     c ^= v & 1; if (c) v += a; c ^= v & 1;
     v ^= b; v <<= 5; i++;
   }
   a -= K[0]; b -= K[1];
}
```

Fig. 5. Simple shift-register block cipher

```
void RC5_ENCRYPT(WORD *pt, WORD *ct)
{
   WORD i, A = pt[0] + S[0], B = pt[1] + S[1];

    for(i = 1; i <= 12; i++)
    {
        A = ROL(A ^ B, B) + S[2*i];
        B = ROL(B ^ A, A) + S[2*i + 1];
    }
    ct[0] = A; ct[1] = B;
}
```

```
rc5(u32 ct[2], u32 S[25])
{
  u32 A, B;
  A <-> ct[0]; B <-> ct[1];
  A += S[0]; B += S[1];

  for (i=1; 13) {
    A ^= B; A <<= B; A += S[2*i];
    B ^= A; B <<= A; B += S[2*i+1];
    i++;
  }
  ct[0] <-> A; ct[1] <-> B;
}
```

Fig. 6. RC5 in C (top) and Hermes (bottom)

whole program to be executed backwards. Since Hermes (like Janus) is designed to be locally reversible, inversion of procedures is simple.

Individual Hermes statements are fairly straightforward to compile to C. The compiler inserts the checks and forced zeroing required for reversibility and compiles statements to time-invariant C. An issue with using C as the target language is that the C compiler may optimise away the statements that clear local variables, hence allowing information to leak. Other optimisations may make timing depend on the actual data, leading to another form of information leak. We are investigating using the zerostack modification of Clang/LLVM [5], which aims to avoid such compiler-introduced leaks.

5 Future Work

To ensure reversibility and avoid information leaks, a number of conditions are tested at run-time: That variables and arrays are zeroed before returning from a procedure, that variables are zero before a scan statement, as well as explicit assertions. We will investigate whether some of these conditions can be verified at compile time, both to reduce the size of the target code, to reduce the running time, and, if all conditions can be verified at compile time, to guarantee that programs will never fail these conditions at run-time.

The speck128 example used a copy of a loop counter variable because loop counters can not be passed as arguments to procedures. We plan to add pro-

cedure parameters that can not be modified inside the procedure, which would allow loop counters to be passed as arguments.

Some of the restrictions for timing-sensitive control can be relaxed if we add variables that are declared to be non-secret. These can, for example, be used for the size of the key or data. Statements that are conditional only on non-secret variables need not be time invariant, as no secret is leaked by this variable timing. This will allow recursion based on, say, the size of data. Loop bound expressions can use non-secret variables, and loop counter variables themselves can be categorised as non-secret. So we plan to add such in future versions of Hermes.

Because if the issues with using C as a target language, we plan to make a compiler to low-level code that ensures that variables are cleared and execution is time invariant. This may use some form of proof-carrying [3] code to allow verification of these properties.

References

1. Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. `https://eprint.iacr.org/2013/404`.
2. C. Lutz. Janus: a time-reversible language. A letter to Landauer. http::://www.tetsuo.jp/ref/janus.pdf, 1986.
3. George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '97, pages 106–119, New York, NY, USA, 1997. ACM.
4. Ronald L. Rivest. The RC5 encryption algorithm. *Dr. Dobb's Journal*, 20(1):146–148, January 1995.
5. L. Simon, D. Chisnall, and R. Anderson. What you get is what you C: Controlling side effects in mainstream C compilers. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 1–15, April 2018.
6. Dominik Táborský, Ken Friis Larsen, and Michael Kirkedal Thomsen. Encryption and reversible computations - work-in-progress paper. In *Reversible Computation - 10th International Conference, RC 2018, Leicester, UK, September 12-14, 2018, Proceedings*, pages 331–338, 2018.
7. David J. Wheeler and Roger M. Needham. TEA, a tiny encryption algorithm. In Bart Preneel, editor, *Fast Software Encryption*, pages 363–366, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.
8. Wikipedia. Rc5. `https://en.wikipedia.org/wiki/RC5`, Accessed February 2019.
9. Wikipedia. Speck (cipher). `https://en.wikipedia.org/wiki/Speck_(cipher)`, Accessed February 2019.
10. Wikipedia. Tiny encryption algorithm. `https://en.wikipedia.org/wiki/Tiny_Encryption_Algorithm`, Accessed January 2019.
11. Tetsuo Yokoyama, Holger Bock Axelsen, and Robert Glück. Principles of a reversible programming language. In *Proceedings of the 5th conference on Computing frontiers*, CF '08, pages 43–54, New York, NY, USA, 2008. ACM.

An Ontology-based Approach to the Agile Requirements Engineering

Marina Murtazina[1][0000-0001-6243-9308] and Tatiana Avdeenko[1][0000-0002-8614-5934]

[1]Novosibirsk State Technical University, 630073 Novosibirsk, Russia
murtazina@corp.nstu.ru

Abstract. The paper presents an approach to the agile requirements engineering based on the OWL ontologies. A brief overview of the benefits of an ontology-based approach to requirements engineering is given. Attention is focused on agile engineering requirements process. The proposed approach uses three interrelated and complementary ontologies. The first ontology is used to represent knowledge about the agile requirements engineering process. The second ontology is designed to match natural language sentences with the requirements in order to identify conflicts. The third ontology is used to accumulate the knowledge about the domain of the software product. The first ontology is core. This ontology consists of classes corresponding to events, roles and artefacts of agile development. Object properties established between the individuals of class can be used to identify directly or indirectly linked requirements and requirements artefacts. This enables maintaining requirements traceability. Also the ontology takes into account particular qualities of working with the requirements in agile development processes including knowledge about the criteria for assessing the quality of user stories being the most common form to record the requirements in agile methods. The ontologies are implemented in the Protégé environment.

Keywords: Requirements engineering, Ontology, Agile Environment.

1 Introduction

The success of software products depends on the extent to which they, as tools, can be effectively used in the implementation of the end user tasks. That is why requirements engineering plays a key role in the software development. The requirements engineering as a field of knowledge includes elicitation, analysis, specification and validation of the requirements [1]. By their nature, the software requirements represent complex knowledge that is extracted in the process of requirements engineering from various sources, including many stakeholders, whose views on the developed product can be diametrically opposed. In this regard, the requirements can be considered as a result of alternative solutions in the field of determining functional and qualitative characteristics of the software.

The decision making process in requirements engineering depends greatly on the experience and intuition of the development team. In particular, the team members

use their experience in analyzing feasibility and determining complexity of the requirements, identifying inconsistencies and incompleteness in the requirements sets, forecasting implementation deadlines, assessing implementation risks, etc. Frequent modifications of the requirements inherent to the agile development methodologies as a reaction to changes in the business environment, requires permanent monitoring on the consistency of the requirements specification and the actual priority of the software product functions. Therefore, the ability to quickly identify and resolve conflicting requirements, and also revise priorities to meet new requirements, is critical to the development of the software development project.

The scientific discourse of recent years is characterized by a focus on the application of ontological models to the software development process [2-4]. Ontologies provide a formal representation of knowledge and relationships between the concepts used in the software development. Ontologies can be used for requirements analysis and specification phases [5]. Ontologies allow to expand the possibilities of model-driven requirements engineering (MDRE) through the use of machine reasoning. Over the past few years ontology-driven requirements engineering (ODRE) has become a leading trend [6].

The success of the ontological approach in the requirements engineering is determined by its capabilities, such as availability of the domain vocabulary, formulating knowledge about the application area and its reuse, understanding the problem area, improving communication between specialists from different fields [7]. Ontologies in the requirements engineering are used to formalize the structure of documents for working with the requirements, to represent the types of the requirements and knowledge about the domain of the software product [8]. Ontologies also allow formulating the rules (axioms) for reasoning about traceability, consistency and completeness of the requirements [9]. The ontological approach is useful for comparing stakeholders' points of view on different subsystems of a single information system [10].

Ontologies can be used to improve the requirements development [11] and requirements management [12] in agile software development process. Ontological approach to the requirements engineering allows to improve the process of user story formulation [13], to facilitate verification of compliance with the quality characteristics of individual user stories and sets of user stories [14], as well as to obtain more accurate assessments of the efforts required to implement user stories [15]. Summarizing the above, it should be noted that most studies of the possibilities of using ontologies in requirements engineering do not take into account the specifics of the project management life cycle or the software development life cycle. To the best of our knowledge, very few papers apply ontology-based approach to the agile project development. However, it seems that the application of the ontology-base approach can be especially valuable for agile development. It is precisely under conditions of permanent changes in the requirements and their priorities inherent to the agile development that knowledge engineering methods prove to be especially useful. Representation of knowledge about the project requirements in the form of ontologies allows the use of machine reasoning methods that can be used for discovering logical inconsistencies.

In the present paper we propose an ontology-based approach to the agile requirements engineering using a system of three ontologies. The first ontology is needed to represent knowledge about the agile requirements engineering process, the second one is designed to match natural language sentences with the requirements in order to identify conflicts, the third ontology is used to accumulate the knowledge about the domain of the software product. The paper is organized as follows. In Section 2 we provide a review of the research topic and the terminology used. In Section 3 we define the ontology for agile requirements engineering process. In section 4, we give conclusions about the prospects of using this approach in agile software development.

2 Theoretical background

2.1 Requirements engineering in agile software development

In agile development, the software requirements specification (SRS) is an integrated requirements package which is maintained up to date. This package is not a single physical document, but a logical structure filled with requirements for a software product.

To date, the Scrum framework is the most popular among the agile project management framework. The basis of Scrum is the theory of empirical control. According to this theory, the source of knowledge is experience, and the source of solutions is real data. The basic scheme of the Scrum framework work is presented in Fig.1.

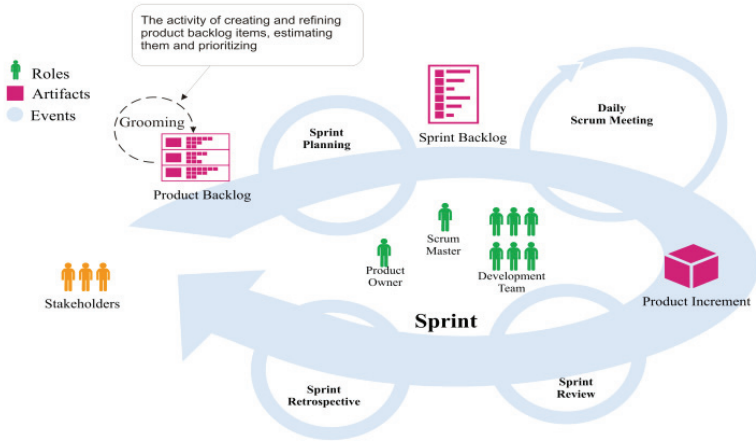


Fig.1. The basic scheme of the Scrum framework work

According to Agile Manifesto [16], agile development encourages the creation of the minimum amount of documentation necessary to manage and coordinate the project participants work in developing a software product. When a software project is launched, it is not necessary to create a comprehensive requirements document. First, the document “Vision and Scope” is developed. This document determines the stake-

holders vision on the software being developed in terms of the key needs and constraints under which the project will be implemented.

Next, the work begins on filling the Product Backlog, which is a prioritized list of currently existing product requirements, which is never complete. Product backlog items can be divided into product features, defect, technical work, knowledge acquisition by type of work [17]. Product feature is a unit of functionality of a software product that satisfies a requirement. The remaining three types of product backlog items are needed to plan the work on eliminating defects, refactoring, database migration, research, necessary to implement the requirements of any type, etc.

Before a feature is scheduled for a sprint, it is necessary to decompose it into small user stories, develop basic behavior scenarios, evaluate implementation efforts, identify dependences on other user stories, and determine the priority. It is also necessary to analyze low priority user stories. Perhaps these product backlog items became important, or, on the contrary, so unimportant that they should be removed. This work is done by the Product Owner together with the development team as a part of the Backlog grooming.

Backlog grooming (product backlog refinement or grooming) is an activity throughout the sprint aimed at revising the backlog of the product to prepare its product for the next sprint planning. Backlog grooming helps ensure that the requirements will be clarified, and user stories will be prepared for work in advance of planning for the sprint. As a result of Backlog grooming, the top of the Product Backlog should include user stories prepared for sprint. In this case, such user stories should be enough for 2-3 sprints. User stories should be clear to all team members, evaluated by the team, and acceptance criteria should be indicated for the stories. The acceptance criteria can be written in the form of simple sentences or behavior scenarios, in particular, in the form of Gherkin scenarios that uses the simple syntax "Given-When-Then".

2.2 User stories

Initially, user stories were recorded on cards of small sizes. The card is not intended to collect all information about the requirement. The card must contain several sentences that reflect the essence of the requirement. The card usually indicates the identifier, name, text, acceptance criteria and its assessment (priority, risk, evaluation of the efforts, etc.). The user story should follow the following pattern:

As a <type of user X >, I want <some goal Y>, So that <some reason Z>

Nevertheless, it is easy to make a lot of mistakes when formulating a user story. Suppose it is necessary to describe the functionality that will allow the social network users to sell stickers sets that they can add to the messages. In this case, the user has a standard free stickers set. Then, suppose the story was formulated as follows:

"As a user, I want to add sticker sets from the paid collection,
So that I can see new stickers in my sticker sets".

In this user story the type of the user is not specified, so there is no understanding what problem the user solves, what is the user motivation. If the users of the system were divided into two types – logged-in users and visitors, then in this user story,

apparently, a logged-in user would be assumed, and, most likely, the developer would immediately understand this and without losing time on figuring out the type of the user will be able to implement the feature correctly. But if the logged-in users were in turn divided into subclasses (for example, there were the users with a premium account for which the service described in the user story should already be included in the payment), the feature might be implemented incorrectly, or the developer would spend time clarifying out what type the user belongs. In order to avoid mistakes in the first part of the user history, it is reasonable to build a domain model of hierarchically related user types, followed by an agreement on the formulation of a user story indicating a specific type of the user from the domain model, and not just "user". Moreover, the above user story inaccurately defines user's action (the second part) and user's motivation (the third part). So it is better to formulate it as follows:

"As a logged-in user, I want to buy new stickers sets,
So that I can decorate my messages with non-standard stickers".

So, formulating the user story is one of the cornerstones of agile engineering requirements. Despite the huge popularity of user stories for agile development, there are only a few numbers of methods to assess their quality. Many existing approaches use the INVEST model proposed in 2003 by Bill Wake. According to this model, user story should be: Independent, Negotiable, Valuable, Estimable, Small, Testable [18].

The development of a user stories list for the project can be preceded by the construction of the Feature tree [19]. Feature tree is a high-level hierarchical diagram illustrating the dependencies between the features of a software product. The Product Owner begins building the Feature tree in Sprint 0. The start of the Feature tree can be generated based on information from the document "Vision and Scope". The Feature tree is constantly evolving since all product features are usually undefined during sprint 0. Further, the items of the Feature tree are added to the Product backlog, where they are supplemented with user stories.

3 OWL ontology for agile requirements engineering

In this paper, it is proposed to use the OWL ontology system to support the requirements engineering. The first ontology contains knowledge of requirements engineering within the framework of an agile approach including knowledge about types of requirements. The second ontology contains knowledge for identifying conflicting requirements. The third one is a domain model that includes a software product feature tree, a user roles model and the connections between them.

In Fig.2 we show the taxonomy of the upper level classes for the ontology "Guide", and also object properties reflecting relations between the ontology classes. The ontology "Guide" consists of classes corresponding to events, roles and artefacts of agile development. The instances (individuals) of the ontology classes are the software requirements and their artefacts, as well as information about the development team and stakeholders. Object properties reflect relations that can be established between individuals. For example, to specify the relationship "the requirement refines another requirement" the object property "refines" is used. This object property, in

turn, includes as subproperties that can be established between different classes (or individuals) of requirements artefacts which are also requirements by their nature (for example, the behavior scenario refines user story). Object property “traceFrom” is intended to define bottom-up tracing links. For the object property “traceFrom” and its subproperties, inverse properties are given through the “Inverse Of” relationship. This allows the top-down tracing of the “traceTo” relationship.

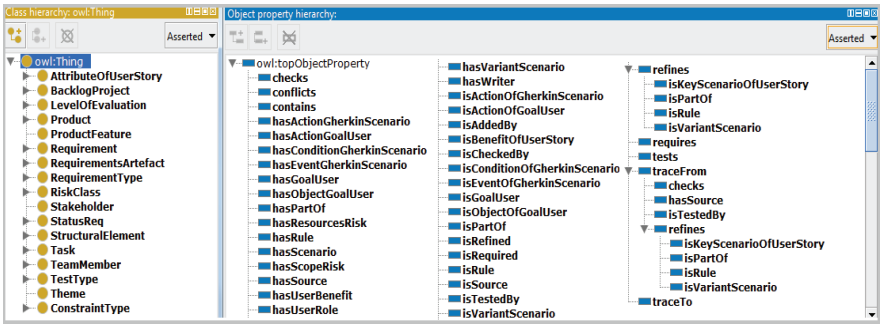


Fig.2. The taxonomy of the upper level classes for ontology “Guide” and object properties

Object property “conflicts” enables specifying that the requirements conflict with each other. This can be done directly in this ontology or transferred from the additional ontology “Detection of conflicts in the requirements”. Fig.3 lists the objects properties for this ontology and their domains and ranges.

Individuals of the ontology “Detection of conflicts in the requirements” are elements extracted from the requirements text. The sentence that expresses a requirement, regardless of the technique used for recording requirements, can be divided into the following main parts: the subject, the action, and the object to which the action of the subject is directed. To identify conflicts between user stories it is necessary to extract the functional user role, the action and the object to which the action is directed, from the user story text. Object properties “sameAsActor”, “sameAsAction” and “sameAsObject” are set between instances of the corresponding classes if the same name is used for the elements of two requirements or the full name in one and an abbreviation in the second. Object properties “isaActor” and “isaObject” are used to establish hierarchical relations. For example, a senior manager is a manager. Object properties “antonymsAction” and “antonymsObject” are set between instances of the corresponding classes if they have the opposite value (for example, “my comment about the product” and “someone else's comment about the product”). Object properties “partOfAction” and “partOfObject” are set between instances of the respective classes if one is part of the other. Object properties “synonymsActor”, “synonymsAction” and “synonymsObject” are set if the values of the corresponding requirements elements have a synonymous value. In all other cases it is considered that the corresponding elements of the requirements are bound by object property “no-relationActor”, “no-relationAction” or “no-relationObject”.

The following production rules are used to determine the type of relations between any two requirements:

If *ObjectProperty_between_class_instances (Actor1, Actor2)*
AND *ObjectProperty_between_class_instances (Action1, Action2)*
AND *ObjectProperty_between_class_instances (Object1, Object2)* Then *Object properties_between_class_instances (Requirement1, Requirement2)*

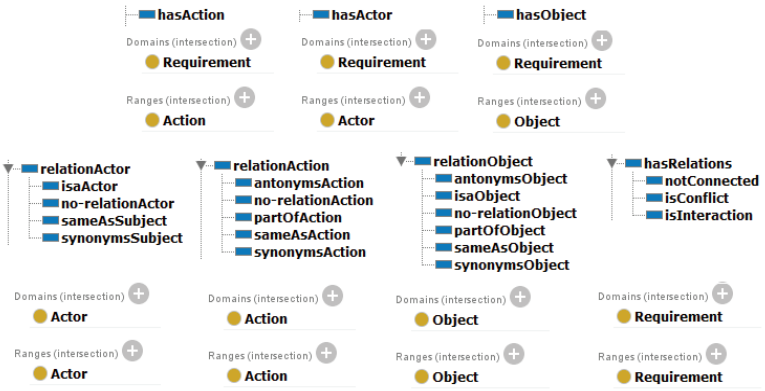


Fig.3. Classes and object properties of the ontology "Detection of conflicts in the requirements"

Relations between actors, actions and objects can be established on the basis of information from the domain ontology as well as using linguistic ontologies, such as WordNet.

To illustrate the idea of the proposed approach to the formation of a software product domain model in the form of an OWL ontology, we consider a fragment of the ontology for an online store (Fig.4).

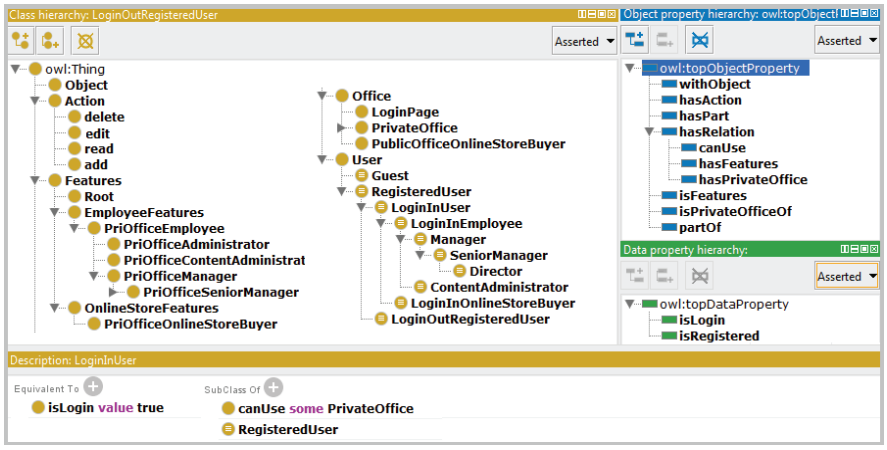


Fig.4. The ontology «OnlineStore»

When building the software product domain ontology it is necessary to analyze the user roles (class "User") and also to decide which software product sections the users

can work with (class “Office”) and to build a Feature tree (class “Features”). Then, it should be determined which sections can be used by certain users, and also indicate which features are associated with these sections. Instances of the class “Object” are objects the user works with (for example, a sales report or a search string). Instances of the class “Action” are verbs that are used to describe user actions. Actions can be divided into four operations: reading, adding, editing, deleting. In this example, users are divided into groups depending on the two properties “isLogin” and “isRegistered”. The knowledge of which office the user has is also used to assign a class to the user.

A list of the features available in the office is indicated for each class of the corresponding office. Subclasses of personal offices automatically inherit features set for the classes to which they belong.

4 Conclusion and future work

The OWL ontology system containing three ontologies was developed based on the analysis of the results of ontology application in requirements engineering. The first ontology accumulates knowledge about the development of software products in an agile environment. The second one contains knowledge about the relations between the elements of sentence with a requirement (role, action, object). This allows analyzing pairs of requirements in order to identify conflicts. The third one contains knowledge of the software product domain.

We have developed a model which enables solving typical problems for requirements engineering in agile software development. These include the formation and refinement of the Product backlog, requirements tracing and the conflicting requirements identification. The next stage of our research will include the development of the algorithm for prioritizing and re-prioritizing requirements based on the business value of the product backlog item, assessment of the efforts to requirements implement and risks, as well as the dependencies between product backlog items that are recorded in the ontology.

Acknowledgments

The reported study was funded by Russian Ministry of Education and Science, according to the research project No. 2.2327.2017/4.6.

References

1. ISO/IEC. Software Engineering - Guide to the software engineering body of knowledge (SWEBOK).2nd edn. ISO/IEC TR 19759 (2015).
2. Lucassen, G., Dalpiaz, F., van der Werf, J.M.E.M., Brinkkemper, S.: Improving agile requirements: the quality user story framework and tool. Requirements Engineering 21.3(2016): 383-403.

3. de Souzam P.L., do Pradom A.F., de Souzam W.L., dos Santos Forghieri Pereiram S.M., Piresm L.F.: Improving Agile Software Development with Domain Ontologies. In: 15th International Conference on Information Technology – New Generations Information Technology-New Generations. pp. 267-274. Springer, Cham (2018).
4. Sitthithanasakul, S., Choosri, N.: Using ontology to enhance requirement engineering in agile software process. In: 2016 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA). IEEE (2016).
5. Bhatia, M. P. S., Kumar, A., Beniwal R.: Ontologies for Software Engineering: Past, Present and Future. Indian Journal of Science and Technology 9(9), http://www.indjst.org/index.php/indjst/article/view/71384/67982, last accessed 2019/02/02.
6. Siegemund, K., Thomas, E. J., Zhao, Y., Pan, J., Assmann U.: Towards Ontology-driven Requirements Engineering. In: Proceedings of the 7th International Workshop on Semantic Web Enabled Software Engineering (SWESE'11), Germany (2011).
7. Valaski, J., Reinehr, S., Malucelli A.: Which Roles Ontologies play on Software Requirements Engineering. In: International Conference on Software Engineering Research and Practice, pp. 24-30, CSREA Press (2016).
8. Castañeda, V., Ballejos, L., Caliusco, M. L., Galli,M.R.: The use of ontologies in requirements engineering. Global Journal of Research In Engineering, 10(6), 2–8 (2010).
9. Goknil, A., Kurtev, I., van den Berg, K.: A Metamodeling Approach for Reasoning about Requirements. In: Schieferdecker, I., Hartman, A. (eds.) Model Driven Architecture – Foundations and Applications. ECMDA-FA 2008. Lecture Notes in Computer Science, vol 5095, pp. 310-325. Springer, Berlin, Heidelberg (2008).
10. Assawamekin, N., Sunetnanta, T., Pluempitiwiriyawej, C.: Ontology-based multiperspective requirements traceability framework. Knowledge and Information Systems, 25(3), 493-522 (2010).
11. Sitthithanasakul S, Choosri N. Using ontology to enhance requirement engineering in agile software process. 2016 10th International Conference on Software, Knowledge, Information Management & Applications, pp. 181-186. IEEE (2017).
12. Avdeenko, T., Murtazina, M.: Intelligent Support of Requirements Management in Agile Environment. In: Borangiu, T., Trentesaux, D., Thomas, A., Cavalieri, S. (eds.) Service Orientation in Holonic and Multi-Agent Manufacturing. SOHOMA 2018. Studies in Computational Intelligence, vol 803, pp. 97-108. Springer, Cham (2019).
13. Thamrongchote, C.,Vatanawood, W.: Business process ontology for defining user story. In: 15th International Conference on Computer and Information Science. IEEE, Okayama, Japan (2016).
14. Murtazina, M.S., Avdeenko, T.V.: Ontology-Based Approach to the Requirements Engineering in Agile Environment. In: Actual Problems of Electronics Instrument Engineering (APEIE) 2018 XIV International Scientific-Technical Conference on, pp. 496-501. IEEE, Novosibirsk, Russia (2018).
15. Adnan, M., Afzal, M.: Ontology Based Multiagent Effort Estimation System for Scrum Agile Method. In: IEEE Access, vol. 5, pp. 25993-26005. IEEE (2017).
16. Agile Manifesto, https://agilemanifesto.org/, last accessed 2019/02/02.
17. Rubin, K. S.: Essential Scrum: A Practical Guide to the Most Popular Agile Process. Addison-Wesley (2013).
18. Wake, B.: INVEST in Good Stories, and SMART Tasks, https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/, last accessed 2019/02/02.
19. Babar, M. A., Brown, A. W., Mistrik I.: Agile Software Architecture: Aligning Agile Processes and Software Architectures (1st ed.). Elsevier, Waltham, MA, USA (2013).

Verification and Validation of Semantic Annotations

Oleksandra Panasiuk, Omar Holzknecht, Umutcan Şimşek, Elias Kärle and Dieter Fensel

University of Innsbruck, Technikerstrasse 21a, Innsbruck 6020, Austria,
firstname.lastname@sti2.at

Abstract. In this paper, we propose a framework to perform verification and validation of semantically annotated data. The annotations, extracted from websites, are verified against the schema.org vocabulary and Domain Specifications to ensure the syntactic correctness and completeness of the annotations. The Domain Specifications allow checking the compliance of annotations against corresponding domain-specific constraints. The validation mechanism will detect errors and inconsistencies between the content of the analyzed schema.org annotations and the content of the web pages where the annotations were found.

Keywords: verification · validation · semantic annotation · schema.org.

1 Introduction

The introduction of the Semantic Web [3] changed the way content, data and services are published and consumed online fundamentally. For the first time, data in websites becomes not only machine-readable, but also machine understand- and interpretable. The semantic description of resources is driving the development of a new generation of applications, like intelligent personal assistants and chatbots, and the development of knowledge graphs and artificial intelligence applications. The use of semantic annotations was accelerated by the introduction of schema.org [8]. Schema.org was launched by the search engines Bing, Google, Yahoo! and Yandex in 2011. It has since become a de-facto standard for annotating data on the web [15]. The schema.org vocabulary, serialized with Microdata, RDFa, or JSON-LD, is used to mark up website content. Schema.org is the most widespread vocabulary on the web, and is used on more than a quarter of web pages [9,14].

Even though studies have shown that the amount of semantically annotated websites are growing rapidly, there are still shortcomings when it comes to the quality of annotations [12,17]. Also the analyses in [10,1] underline the inconsistencies and syntactic and semantic errors in semantic annotations. The lack of completeness and correctness of the semantic annotations makes content unreachable for automated agents, causes incorrect appearances in knowledge graphs and search results, or makes crawling and reasoning less effective for

building applications on top of semantic annotations. These errors may be caused by missing guidelines, insufficient expertise and technical or human errors. Data quality is a critical aspect for efficient knowledge representation and processing. Therefore, it is important to define methods and techniques for semantic data verification and validation, and to develop tools which will make this process efficient, tangible and understandable, also for non-technical users.

In this paper, we extend our previous work [21], where we introduced a Domain Specification, and present an approach for verification and validation of semantic annotations. A Domain Specification (DS) is a design pattern for semantic annotations; an extended subset of types, properties, and ranges from schema.org. The semantify.it Evaluator[1]is a developed tool that allows the verification and validation of schema.org annotations which are collected from web pages. Those annotations can be verified against the schema.org vocabulary and Domain Specifications. The verification against Domain Specifications allows for the checking of the compliance of annotations against corresponding domain-specific constraints. The validation approach extends the functionality of the tool by detecting the consistency errors between semantic annotations and annotated content.

The remainder of this paper is structured as follows: Section 2 describes the verification approach of semantic annotations. Section 3 describes the validation approach. Section 4 concludes our work and describes future work.

2 Verification

In this section we discuss the verification process of semantic annotations according to schema.org and Domain Specifications. The section is structured as follows: Section 2.1 gives the definition of the semantic annotation verification, Section 2.2 describes related work, section 2.3 discusses our approach, and Section 2.4 describes the evaluation method.

2.1 Definition

The verification process of semantic annotations consists of two parts, namely, (I) checking the conformance with the schema.org vocabulary, and (II) checking the compliance with an appropriate Domain Specification. While the first verification step ensures that the annotation uses proper vocabulary terms defined in schema.org and its extensions, the second step ensures that the annotation is in compliance with the domain-specific constraints defined in a corresponding DS.

2.2 Related Work

In this section, we refer to the existing approaches and tools to verify structured data. There are tools for verifying schema.org annotations, such as the

[1] `https://semantify.it/evaluator`

Google Structured Data Testing tool[2], the Google Email Markup Tester[3], the Yandex Structured Data Validator[4], and the Bing Markup Validator [5]. They verify annotations of web pages that use Microdata, Microformats, RDFa, or JSON-LD as markup formats against schema.org. But these tools do not provide the check of completeness and correctness. For example, they can allow one to have empty range values, redundancy of information, or semantic consistency issues (e.g. the end day of the event is earlier than the start day). In [7] SPARQL and SPIN are used for constraint formulation and data quality check. The use of SPARQL and SPIN query template sets allows the identification of syntax errors, missing values, unique value violations, out of range values, and functional dependency violations. The Shape Expression (ShEx) definition language [20] allows RDF verification[6] through the declaration of constraints. In [4] authors define a schema formalism for describing the topology of an RDF graph that uses regular bag expressions (RBEs) to define constraints. In [5] the authors described the semantics of Shapes Schemas for RDF, and presented two algorithms for the verification of an RDF graph against a Shapes Schema. The Shapes Constraint Language[7] (SHACL) is a language for formulating structural constraints on RDF graphs. SHACL allows us to define constraints targeting specific nodes in a data graph based on their type, identifier, or a SPARQL query. The existing approaches can be adapted for our needs but not fully, as they are developed for RDF graph verification and not for schema.org annotations in particular.

2.3 Our approach

To enable the verification of semantic annotations according to the schema.org vocabulary and to Domain Specifications, we developed a tool that executes a corresponding verification algorithm. This tool takes as inputs the schema.org annotation to verify and a DS that corresponds to the domain of the annotation. The outcome of this verification process is provided in a formalized, structured format, to enable the further machine processing of the verification result.

The verification algorithm consists of two parts, the first checks the general compliance of the input annotation with the schema.org vocabulary, while the latter checks the domain-specific compliance of the input annotation with the given Domain Specification. The following objectives are given for the conformity verification of the input annotation according to the schema.org vocabulary:

1. The correct usage of serialization formats allowed by schema.org, hence RDFa, Microdata, or JSON-LD.
2. The correct usage of vocabulary terms from schema.org in the annotations, including types, properties, enumerations, and literals (data types).

[2] `https://search.google.com/structured-data/testing-tool/`
[3] `https://www.google.com/webmasters/markup-tester/`
[4] `https://webmaster.yandex.com/tools/microtest/`
[5] `https://www.bing.com/toolbox/markup-validator`
[6] Authors use term "validation" in their paper due to content definition.
[7] `https://www.w3.org/TR/shacl-ucr/`

3. The correct usage of vocabulary relationships from schema.org in the annotations, hence, the compliance with domain and range definitions for properties.

The domain-specific verification of the input annotation is enabled through the use of Domain Specifications[8], e.g. DS for annotation of tourism domain and GeoData [18,19]. Domain Specifications have a standardized data model. This data model consists of the possible specification nodes with corresponding attributes that can be used to create a DS document (e.g. specification nodes for types, properties, ranges, etc.). A DS document is constructed by the recursive selection of these grammar nodes, which, as a result, form a specific syntax (structure) that has to be satisfied by the verified annotations [11]. Keywords in these specification nodes allow the definition of additional constraints (e.g. "multipleValuesAllowed" or "isOptional" for property nodes). In our approach, the verification algorithm has to ensure that the input annotation is in compliance with the domain-specific constraints defined by the input DS. In order to achieve this, the verification tool has to be able to understand the DS data model, the possible constraint definitions, and to check if verified annotations are in compliance with them.

2.4 Evaluation

We implement our approach in the semantify.it Evaluator[9]. The tool provides a verification report with detailed information about detected errors according to the schema.org vocabulary (see Fig.1) and Domain Specifications (see Fig.2).

Nr.	Type	Markup	View	SDO-Valid
1	MusicEvent	jsonld		Valid with Warnings
2	WebSite	jsonld		Valid with Warnings
3	BreadcrumbList	jsonld		Not Valid
4	BreadcrumbList	microdata		Valid
5	PostalAddress	microdata		Valid

Fig. 1. Schema.org Verification

Besides the verification result itself, the report includes details about the detected errors, e.g. error codes (ID of the error type), error titles, error severity levels, error paths (where within the annotation the error occurred), and textual descriptions of the errors. The implementation itself can be evaluated

[8] List of available Domain Specifications: `https://semantify.it/domainSpecifications/public`

[9] `https://semantify.it/evaluator`

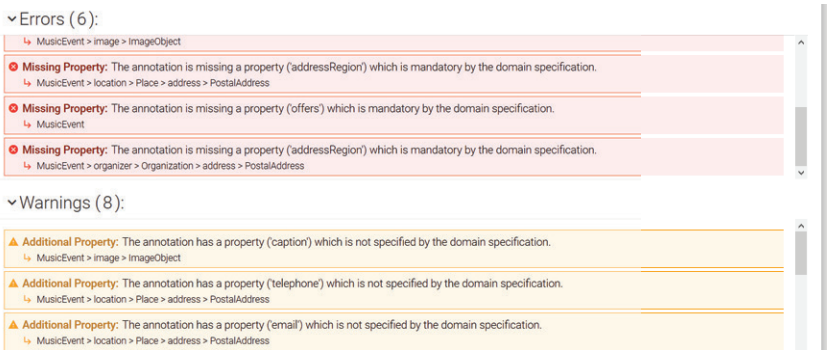


Fig. 2. Domain Specification Verification. Verification Report

through unit tests in terms of a correct functionality (correctness) and the implementation of all possible constraint possibilities of the Domain Specification vocabulary (completeness). This can be achieved by comparing the structured representation of the result, namely the JSON file produced by the verification algorithm, which is used to generate a human-readable verification report for the user (see Fig.3), with the expected verification report outcome specified in the test cases for predefined annotation-Domain Specification pairs.

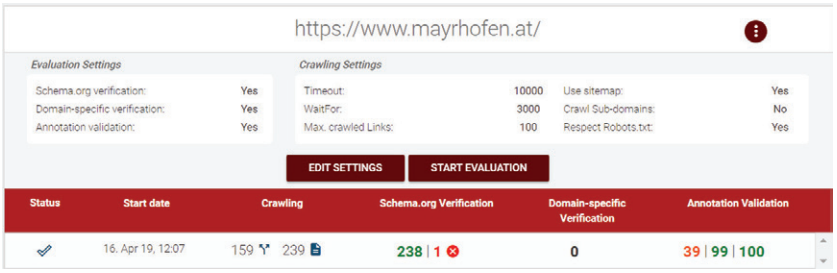


Fig. 3. semantify.it Evaluator. Verification and Validation Report

A formal proof of the correctness and completeness of our implemented algorithm is rather straight forward given the simplicity of our current knowledge representation formalism. In our ongoing work[10], we develop a richer constraint language which will require more detailed analysis of these issues.

3 Validation

Search engines may penalize the publisher of structured data if their annotations include content that is invisible to users, and/or markup irrelevant or mislead-

[10] The paper is under double blind review and can't be revealed

ing content. These penalties may have negative effects on a website (e.g. bad position of the website in search results) or even lead to a non-integration of the structured data (e.g. no generation of rich snippets). For example, annotations of the Destination Management Organizations (DMOs) usually include a list of offers. These offers must comply with offers which are described on the website, and all URLs contained in the annotations must match with the URLs in the content. Such issues can be detected through the validation of semantic annotations.

In this section, we discuss the validation process of semantic annotations and the proposed approach. The section is structured as follows: Section 3.1 gives the definition of the semantic annotation validation, Section 3.2 describes some related work, Section 3.3 discusses our approach, and Section 3.4 describes the evaluation method.

3.1 Definition

The validation of semantic annotations is the process of checking whether the content of a semantic annotation corresponds to the content of the web page that it represents, and if it is consistent with it. Semantic annotations should include the actual information of the web page, correct links, images and literal values without overlapping or redundancy.

3.2 Related Work

The incorrect representation of the structured data can make data unreachable for automated engines, cause an incorrect appearance in the search results, or make crawling and reasoning less effective for building applications on top of semantic data. The errors may be caused by not following recommended guidelines, e.g. structured data guidelines[11], insufficient expertise, technical or human errors (some of the issues can be detected by Google search console[12]), and/or annotations not being in accordance with the content of web pages, so-called "spammy structured markup"[13]. There is no direct literature related to the methods of detecting inconsistency between semantic annotations and content of web pages, but the problem of the content conformity restriction is also mentioned in [13].

3.3 Our approach

Since semantic annotations are created and published by different data providers or agencies in varying quantity and quality and using different assumptions, the validity of data should be prioritized to increase the quality of structured data. To solve the problem of detecting errors caused by inconsistencies between analyzed

[11] https://developers.google.com/search/docs/guides/sd-policies
[12] https://search.google.com/search-console/about
[13] https://support.google.com/webmasters/answer/9044175?hl=en&visit_id=636862521420978682-2839371720&rd=1#spammy-structured-markup

schema.org annotations and the content of the web pages where the annotations were found, we propose a validation framework. The framework consists of the following objectives:

1. Detect the main inconsistencies between the content of schema.org annotations and the content of their corresponding web pages.
2. Develop an algorithm for the consistency check between a web page and corresponding semantic annotations. The information from web pages can be extracted from the source of a web page by tracking the appropriate HTML tags, keywords, lists, images, URLs, paragraph tags and the associated full text. Some natural language processing and machine learning techniques can be applied to extract important information from the textual description, e.g price, email, telephone number and so on. There exist some approaches, such as named entity recognition [16] to locate and categorize important nouns and proper nouns in a text, web information extraction systems [6], text mining techniques [2].
3. Define metrics to evaluate the consistencies of the semantic annotations according to the annotated content. In this step, we analyze existing data quality metrics that can be applied on the structured data and define metrics that can be useful to evaluate the consistency between a web page content and semantic annotation. We measure the consistency for different types of values, such as URL, string, boolean, enumeration, rating value, date and time formats.
4. Provide a validation tool to present the overall score for a web page and detailed insights about the evaluated consistency scores on a per value level.

3.4 Evaluation

To ensure the validity of the report results, we will organize a user study of semantic annotations and annotated web pages to prove the performance of our framework. The questionnaire will be structured in a way to get quantitative and qualitative feedback about the consistencies between a web page and annotation content (see Fig. 4) according to the results provided by the framework (see Fig.3). As our use case, we will use annotated data and websites of Destination Management Organizations, such as Best of Zillertal Fügen[14], Mayrhofen[15], Seefeld[16], and Zillertal Arena[17].

4 Conclusion and Future Work

Semantic annotations will be used for improved search results by search engines or as building blocks of knowledge graphs. Therefore, the quality issues in terms

[14] `https://www.best-of-zillertal.at`
[15] `https://www.mayrhofen.at`
[16] `https://www.seefeld.com/`
[17] `https://www.zillertalarena.com`

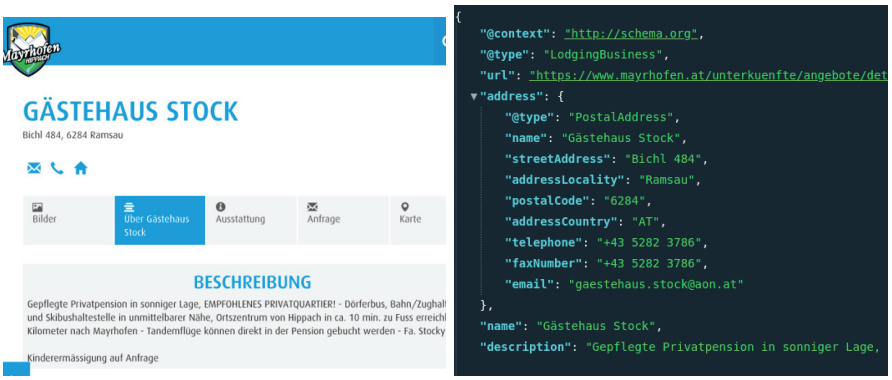


Fig. 4. Web page content and annotation content

of structure and consistency can have an impact on where the annotations are utilized and lead, for instance, to false representation in the search results or to low-quality knowledge graphs. In this paper, we described our ongoing work for an approach to verify and validate semantic annotations and the tool that is evolving as the implementation of this approach.

For the future work, we will define Domain Specifications with SHACL in order to comply with the recent W3C Recommendation for RDF validation. We will develop an abstract syntax and formal semantics for Domain Specifications and map it to SHACL notions, for instance by aligning the concept of Domain Specifications with SHACL node shapes.

References

1. Akbar, Z., Kärle, E., Panasiuk, O., Şimşek, U., Toma, I., Fensel, D.: Complete semantics to empower touristic service providers. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". pp. 353–370. Springer (2017)
2. Allahyari, M., Pouriyeh, S., Assefi, M., Safaei, S., Trippe, E.D., Gutierrez, J.B., Kochut, K.: A brief survey of text mining: Classification, clustering and extraction techniques. arXiv preprint arXiv:1707.02919 (2017)
3. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific american **284**(5), 34–43 (2001)
4. Boneva, I., Gayo, J.E.L., Hym, S., Prudhommeau, E.G., Solbrig, H.R., Staworko, S.: Validating rdf with shape expressions. CoRR, abs/1404.1270 (2014)
5. Boneva, I., Gayo, J.E.L., Prudhommeaux, E.G.: Semantics and validation of shapes schemas for rdf. In: International Semantic Web Conference. pp. 104–120. Springer (2017)
6. Chang, C.H., Kayed, M., Girgis, M.R., Shaalan, K.F.: A survey of web information extraction systems. IEEE transactions on knowledge and data engineering **18**(10), 1411–1428 (2006)

7. Fürber, C., Hepp, M.: Using sparql and spin for data quality management on the semantic web. In: International Conference on Business Information Systems. pp. 35–46. Springer (2010)
8. Guha, R.: Introducing schema. org: Search engines come together for a richer web. Google Official Blog (2011)
9. Guha, R.V., Brickley, D., Macbeth, S.: Schema. org: Evolution of structured data on the web. Communications of the ACM **59**(2), 44–51 (2016)
10. Hollenstein, N., Schneider, N., Webber, B.L.: Inconsistency detection in semantic annotation. In: LREC (2016)
11. Holzknecht, O.: Enabling Domain-Specific Validation of Schema.org Annotations. Master's thesis, Innsbruck University, Innrain 52, 6020 Innsbruck, Austria (Nov 2018)
12. Kärle, E., Fensel, A., Toma, I., Fensel, D.: Why are there more hotels in tyrol than in austria? analyzing schema. org usage in the hotel domain. In: Information and Communication Technologies in Tourism 2016, pp. 99–112. Springer (2016)
13. Kärle, E., Fensel, D.: Heuristics for publishing dynamic content as structured data with schema. org. arXiv preprint arXiv:1808.06012 (2018)
14. Meusel, R., Petrovski, P., Bizer, C.: The webdatacommons microdata, rdfa and microformat dataset series. In: International Semantic Web Conference. pp. 277–292. Springer (2014)
15. Mika, P.: On schema. org and why it matters for the web. IEEE Internet Computing **19**(4), 52–55 (2015)
16. Mohit, B.: Named entity recognition. In: Natural language processing of semitic languages, pp. 221–245. Springer (2014)
17. Mühleisen, H., Bizer, C.: Web data commons-extracting structured data from two large web corpora. LDOW **937**, 133–145 (2012)
18. Panasiuk, O., Kärle, E., Şimşek, U., Fensel, D.: Defining tourism domains for semantic annotation of web content. e-Review of Tourism Research **9** (Jan 2018), research notes from the ENTER 2018 Conference on ICT in Tourism
19. Panasiuk, O., Akbar, Z., Gerrier, T., Fensel, D.: Representing geodata for tourism with schema.org. In: Proceedings of the 4th International Conference on Geographical Information Systems Theory, Applications and Management - Volume 1: GISTAM,. pp. 239–246. INSTICC, SciTePress (2018)
20. Prud'hommeaux, E., Labra Gayo, J.E., Solbrig, H.: Shape expressions: an rdf validation and transformation language. In: Proceedings of the 10th International Conference on Semantic Systems. pp. 32–40. ACM (2014)
21. Şimşek, U., Kärle, E., Holzknecht, O., Fensel, D.: Domain specific semantic validation of schema. org annotations. In: International Andrei Ershov Memorial Conference on Perspectives of System Informatics. pp. 417–429. Springer (2017)

Improvement of Firebrand Tracking and Detection Software

Prohanov S.[1[0000-0002-4478-2249]], Kasymov D.[1,*[0000-0003-3449-788X]], Zakharov O.[1], Agafontsev M.[1[0000-0003-1383-7291]], Perminov V.[1[0000-0001-5393-1851]], Martynov P.[1[0000-0001-9489-1390]], Reyno V.[2], Filkov A.[3[0000-0001-5927-9083]]

[1] Tomsk State University, Tomsk, Russia
[2] V.E. Zuev Institute of Atmospheric Optics of the Siberian Branch of the Russian Academy of Science, Tomsk, Russia
[3] The University of Melbourne, Creswick, Australia
denkasymov@gmail.com

Abstract. Burning and glowing firebrands generated by wildland and urban fires may lead to the initiation of spot fires and the ignition of structures. One of the ways to obtain this information is to process thermal video files. Earlier, a number of algorithms were developed for the analysis of the characteristics of firebrands under field conditions. However, they had certain disadvantages. In this regard, this work is devoted to the development of new algorithms and their testing.

For this purpose, semi-field experiments were conducted using an apparatus for generating firebrands to obtain the necessary thermal video files. The thermograms were processed to create an annotated IR video base that was further used to test the detector and the tracker.

To detect firebrands in the thermograms, the Laplacian of Gaussian and Difference of Gaussians (DoG) algorithms were tested. To estimate the accuracy of detectors, an original approach involving the application of the F1 score was used. The analysis showed that both algorithms can provide the necessary accuracy for the detection of firebrands and are comparable in time and accuracy, but the DoG algorithm is easier controlled and implemented.

Different firebrand tracking algorithms have been developed and tested. In particular, a Hungarian algorithm-based tracker that can track firebrands between frames with high accuracy is implemented. The comparison of the algorithms showed that Hungarian algorithm-based trackers more accurately tracked the movement of particles.

Keywords: Algorithm, Detection, Firebrands, Tracking.

1 Introduction

In recent years, the number of wildland urban interface fires (WUI fires) has increased. The ignition of buildings in WUI areas is a serious international problem due to large fires in Australia, Greece, Portugal, Spain and the USA [1-4]. The main factors affecting the ignition of building materials and the spread of such fires are radia-

tion and convective heat transfer, as well as firebrands that can accumulate on the roof and in the corners of buildings, on fences or find another way to penetrate into buildings and cause a fire [5-11].

In addition, burning and glowing firebrands produced in the fire front can be transported by wind and cause spot fires. WUI fires are expected to be a serious problem not only for the USA, Europe and Australia, but also for Russia.

At present there is a need in a quantitative understanding of the short distance spotting dynamics, namely the firebrand distribution within a distance from the fire front and how fires coalesce. The absence of such data is an obstacle to the development of fire hazard forecasting methods, as well as to the improvement of measures and recommendations for more efficient and effective work to prevent fires, control and extinguish ground forest fires in proximity to residential buildings.

To address this, a first version of custom software was developed in order to detect the location and the number of flying firebrands in a thermal image and then determine the temperature and sizes of each firebrand [12]. The software consists of two modules, the detector and the tracker. The detector determines the location of firebrands in the frame, and the tracker compares a firebrand in different frames and determines the identification number of each firebrand. However, used algorithms had certain disadvantages. In this regard, this work is devoted to the development of new algorithms and their testing using experimental data.

The non-contact IR diagnostic method based on modern high-speed IR cameras with high spatial resolution can estimate the temperature and size of particles, as well as the speed and trajectory of their movement. These data and the setup simulating the transfer of burning and glowing firebrands depending on the speed of air and their number [13-15] allow the software to be adjusted and verified.

2 Methods

2.1 Creation of an annotated video database

In 2015, a unique setup was designed and built to produce burning and glowing firebrands of various types, sizes, speeds and shapes [13, 16] (see Fig. 1).

Fig. 1. Setup for producing burning and glowing firebrands.

This setup designed by the authors has a number of distinctive features. A screw conveyer with an air smoke screen was mounted in the setup for a long continuous supply of fuels. The setup consists of three units, and its characteristics can be easily changed for different tasks [16]. The setup generating burning and glowing firebrands includes a unit for measuring temperatures and heat fluxes, a video camera and high-speed IR cameras (JADE J530SB and FLIR X6530sc) used as recording equipment for tracking of firebrands (see Fig. 2).

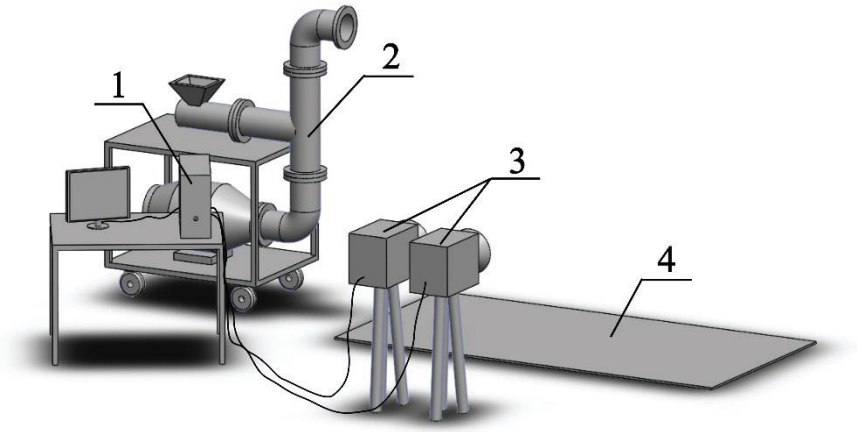


Fig. 2. Experimental setup: 1 is the data recording system, 2 is the producer of burning and glowing firebrands, 3 is the IR cameras, 4 is the test site.

The important characteristics of firebrands produced by wildland fires are the temperature, the distance of transport, and the flight of trajectory. IR cameras in combination with the firebrand producing setup successfully determine the desired characteristics.

A thermal imaging camera (JADE J530SB) containing a narrow-band optical filter with a 2.5 - 2.7 μm spectral interval and measuring the temperature in the range of 310 - 1500 C was used to determine the temperature of firebrands generated by the setup.

The thermal imaging camera had a matrix with a resolution of 320x240 pixels. A lens with a focal length of 50 mm was used for recording; the recording rate was 50 Hz. The optical filter and the lens had the factory calibration. The distance from the output of the particle generator to the thermal imaging camera was 8.7 m. The FLIR X6530sc thermal imaging camera with a spectral interval of 1.5 - 5.1 μm was used to determine the geometrical parameters of flying firebrands. The thermal imaging camera had a matrix with a resolution of 640x512 pixels. A lens with a focal length of 25 mm was used for recording; the recording rate was 50 Hz. The distance from the thermal imaging camera to the central plane of the output of the setup was 2.2 m, and the distance between the thermal imaging cameras was 0.4 m. The measured area was 1.7x1.35 m for the FLIR X6530sc and 0.42x0.32 m for the JADE J530SB.

Natural particles (pine bark and twigs) and wood pellets were used as firebrands. The size of the particles was selected in accordance with the data of field experiments which showed that particles of similar sizes prevailed during a surface fire in a pine forest [17–19]. Natural particles were made of pine bark (Pinus sibirica) and were 10×10, 15×15, 20×20, 25×25, 30×30 mm^2 in size and 5 mm in thickness. Pine twigs with a diameter of 2–4, 4–6, 6–8 and a length of 10, 20, 40 and 60 mm were also used. The diameter of wood pellets was 8 mm, the length of granules was varied from 30 to 50 mm. Each experiment had at least 3 repetitions.

2.2 Development of GUI

A graphical user interface (GUI) was developed to work with a software for detecting, tracking, and determining the characteristics of burning and glowing firebrands in a video using thermal imaging cameras.

To develop the GUI, available libraries of graphic elements that support working with the used software language python3 were considered and compared. In particular, the libraries Tkinter [20], pyGTK / PyGObject [21], wxPython [22], and pyQt5 [23] were considered.

Tkinter is a cross-platform library for developing a graphical interface, stands for the Tk interface and is an interface to Tcl/tk [24], is included in the Python language and does not require additional software installation. However, this library has a relatively small set of built-in widgets and rather poor integration with the desktop environment. The PyGTK/PyGObject library is used to develop cross-platform applications for GNU/Linux and Windows operating systems and write a quite compact code, but it requires additional software installation and the interface of the application under Windows OS is quite non-native. The wxPython library is the wrapper of the wxWidgets library [25]. It also can be used to develop cross-platform applications, but this library is quite complicated and has incomplete documentation. The choice was made in favor of the PyQt5 library, since it can work with the Qt cross-

platform framework [26] which allows applications to be run under various operating systems (Windows, MacOS X, Linux).

To render the video frames, a free graphic component library Qwt [27] that works in conjunction with a library Qt was used. This library provides animation and scaling of data.

The graphical interface is used to open and visualize videos in the ASCII frame format.

Example of the used data format (ASCII):

```
General information :
File  G:\DRAGON_11_05_2018\Capture004_comp.ptw
Date  Sunday, March 05, 2018
Total frames 2789
Format  320 240
Radiometric data :
Calibration file G:\DRAGON_11_05_2018\Capture004_comp.exp
Unit  °C
Emissivity  0.95
BackGround temperature 22.00 °C
Transmission  100.00  %
Distance  2200.00 mm
Atmosphere temperature 26.00 °C
Housing temperature  22.70 °C
Pixel size  25.00 µm
Pixel pitch 30.00 µm
Focal length  50.00 mm
Aperture  2.00  F/#
Cut on  3.70  µm
Cut off 4.80  µm

Image Data :
Frame 0
Time  02:53:41,417
280.01  280.93  280.62  280.97  280.49  280.71  280.32
  281.01  280.49 …
…
282.10  282.32  282.40  282.36  282.32  282.71  282.32
  282.58  282.58 …

Image Data :
Frame 1
Time  02:53:41,437
…
```

2.3 Detector testing

To detect burning and glowing firebrands on the frame, various Gaussian convolution algorithms were tested. The algorithms are aimed at identifying pronounced areas. The Laplacian of Gaussian (LoG) and Difference of Gaussian (DoG) algorithms were tested.

The Gaussian of Laplacian algorithm (LoG) is based on the convolution (filtering) of an image using the Laplace operator:

$$G_\sigma(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(\frac{x^2+y^2}{2\sigma^2}\right)}, \quad (1)$$

where G is the Gaussian kernel, σ is the Gaussian parameter, x, y are spatial coordinates.

The Laplacian of Gaussian can be given by the formula

$$\Delta\left[\left(G_\sigma(x,y) * f(x,y)\right)\right] = LoG * f(x,y), \quad (2)$$

where $LoG = \frac{d^2}{dx^2} G_\sigma(x,y) + \frac{d^2}{dy^2} G_\sigma(x,y)$ is the Laplacian operator.

The Difference of Gaussian (DoG) algorithm is based on the two convolutions of the image with a Gaussian with a different parameter of σ.

$$g_{\sigma_1}(x,y) = G_{\sigma_1}(x,y) * f(x,y), \quad (3)$$

$$g_{\sigma_2}(x,y) = G_{\sigma_2}(x,y) * f(x,y), \quad (4)$$

where $G_{\sigma_1}(x,y)$ and $G_{\sigma_2}(x,y)$ are the Gaussian kernels.

The second step of the algorithm is pixel-by-pixel subtraction of the Gaussian images from each other.

$$g_{\sigma_1}(x,y) - g_{\sigma_2}(x,y) = DoG * f(x,y), \quad (5)$$

where $DoG = G_{\sigma_1} - G_{\sigma_2}$ is the Difference of Gaussian operator.

To evaluate the accuracy of the detectors, the F1 score was used [28]:

$$\mathrm{F1} = \frac{2\mathrm{TP}}{2\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}, \quad (6)$$

where TP is the number of true positives, FP is the number of false positives and FN is the number of false negatives.

This metric takes into account two types of false positives: (i) when a background element is falsely detected (false operation), (ii) when a visible particle is not detected.

2.4 Tracker Testing

After detecting all particles in the frame, it is necessary to determine whether it is a new particle or it is in the previous frame and assign it a unique identification number as well. For this purpose, special particle trackers were developed.

In the previous version of the software, the tracker was based on the nearest neighbor search method (NSS) [29]. Each detection in the current frame, according to the selected metric, was compared to the detection in the next frame, all other detections were ignored. The advantages of this tracker are simple implementation, high operation speed, low memory consumption. However, the method has significant disadvantages, such as a high error, a strong dependence on the choice of the metric and, as a consequence, a low accuracy of operation.

Therefore, additional trackers were tested. A tracker based on the Hungarian algorithm [30] (Kuhn-Munkres algorithm) can track the movement of detection between frames with a high accuracy. The algorithm is applied to all pairs of frames; all detections in the first frame are compared to all detections in the next frame (except the absence of corresponding detection). The main stage of the algorithm is the construction of a cost matrix that, according to the algorithm, can find the optimal match between the considered detections. Unlike the nearest neighbor search algorithm, this algorithm compares all detections of particles between adjacent frames with each other and determines the optimal match, rather than compares the detections individually. This allows the algorithm to work with a higher accuracy. The distance between particles, the size and temperature of the particle were used as a metric for comparing particles in frames.

The Multiple Object Tracking Precision (MOTP) and Multiple Object Target Accuracy (MOTA) metrics [31] were used to evaluate the quality of the trackers.

The MOTP metric is calculated by the formula (7) and used to evaluate the positioning accuracy of the tracked particles.

$$MOTP = \frac{\sum_{i,t} d_i^t}{\sum_t c_t}, \tag{7}$$

where d_i^t is the distance between the ground truth, i.e. annotated particles and the particles predicted by the code with number *i* in the frame; c_t is the number of matches found in the frame with number *t*. This metric strongly depends on the accuracy of the detector.

The MOTA metric is based on the frequency of false positives, the frequency of missed detections and the frequency of errors in assigning a detection number.

$$MOTA = 1 - \frac{\sum_t (m_t + fp_t + mme_t)}{\sum_t g_t}, \tag{8}$$

where m_t is the number of missed detections, fp_t is the number of false positives, mme_t is the number of mismatches; g_t is the number of objects in the frame *t*. The index *t* is responsible for the time or number of the frame.

3 Results and discussion

3.1 GUI

The developed graphic interface (see Fig. 3) is used to navigate through video frames (forward, backward, choice of the frame with a required number), enlarge the window with frames by a given or arbitrary number of times using the mouse, as well as to select and adjust the parameters of the particle tracking algorithm, run the selected tracker with the given parameters, and generate the video in AVI format using the tracker.

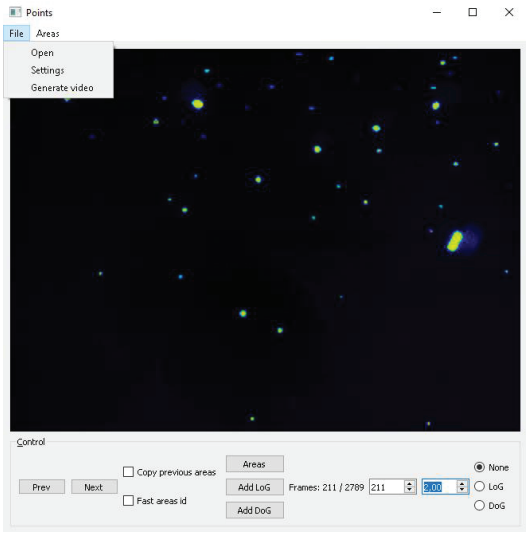


Fig. 3. Graphical user interface.

3.2 Video annotation software

To check the quality of the developed detectors and trackers, there is a need in reference information on the location and trajectory of the flying firebrands. Such information is usually supplied to the input as a special file, in which all detections and tracks of particles are marked and numbered for all frames of the video. Data are usually marked manually. This process is rather long and time-consuming, since a large number of frames have to be marked to test the algorithms. The specialized software accelerates this process by providing the user with quite convenient and effective means for marking data.

For this purpose, an annotation video software was developed and integrated into the main graphical software interface (see Fig. 4). It is used to mark frames in both manual and semi-automatic mode.

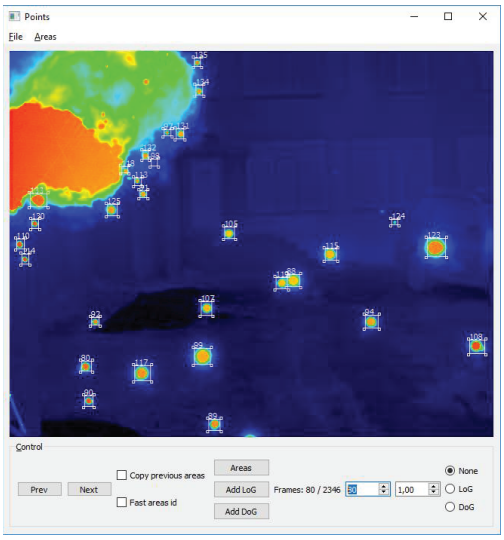


Fig. 4. Frame marked using the annotation video software.

Manual marking is carried out drawing a bounding box around the detection using a mouse. This rectangle has to be assigned a particle number (track number). Rectangles can be moved, copied, deleted and resized.

To assist in annotating video, checkboxes “Copy previous area” and “Fast areas id” are provided in the graphical interface. The first checkbox copies all bounding boxes in the previous frame when moving to the new frame, and the second checkbox automatically numbers the new detections (each new detection will be assigned a number that is larger than the previous number by one).

The interface also includes the “Add LoG” and “Add DoG” buttons which automatically run available particle tracking algorithms (LoG or DoG) in the current frame. If necessary, the operation of automatic detectors can be corrected manually, for example, deleting or adding a new detection, and correcting the number of the particle track. A video can be automatically marked using built-in detectors and trackers.

For quick access to the track by number, there is the “Areas” menu that is used to get or change the coordinates of the bounding box by number. Also, this menu deletes the false detection by its number.

To save and load marked data, a file in the json-format is used [32], in which the frame number, the center coordinates, and the length and width of all bounding boxes marked in it are stored. All detections have unique numbers corresponding to the track number.

Example of the video annotation file:

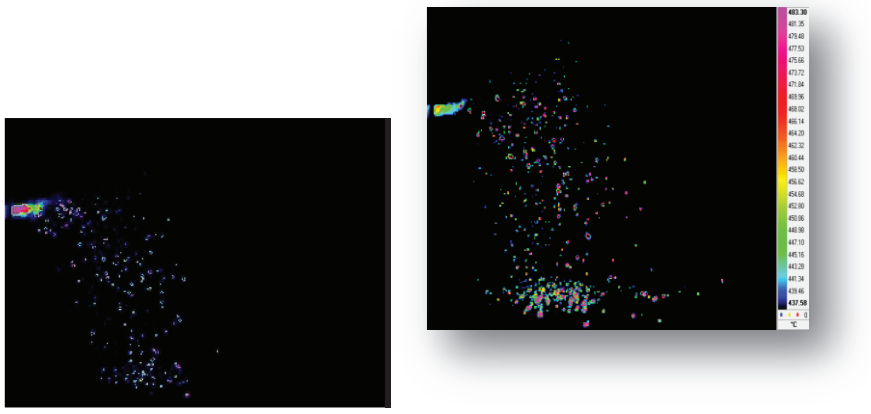
```
{
   "annotations": [
      {
```

```
        "frame_id": 1,
        "data": [
            {
                "id": "1",
                "bbox": [
                    57,
                    239,
                    22,
                    20
                ]
            },
...
```


The Python3 software language was used to develop the software. To create an application, the PyInstaller program [33] was used, which bundles the Python application and all the necessary dependencies into a single package. Thus, a user can run the software without installing any additional modules in the system.

3.3 Annotated video database

A series of semi-field experiments on thermal mapping of the generation and transport of burning and glowing firebrands were conducted using high-speed infrared cameras. 15 videos of the generation and transport of particles were recorded. Each video is a set of thermograms/frames (see Fig. 5). The time of each experiment ranged from 40 to 80 seconds.



a b

Fig. 5. Example of recorded thermograms/frames (infrared): (a) pine twigs; (b) pellets.

Table with the characteristics of recorded videos is given below.

Table 1. Video database.

File #	Weight of particles, g	Number of particles, pcs	Fuel type	JADE J530SB		FLIR X6530sc	
				Number of frames	Time of recording, s	Number of frames	Time of recording, s
1	---	---	bark/twigs	3125	62	---	---
2	---	---	twigs	3048	61	---	---
3	---	---	twigs	3356	67	---	---
4	---	---	twigs	2789	56	---	---
5	---	---	twigs	2220	44	---	---
6	---	---	twigs	3728	78	---	---
7	---	---	twigs	3978	80	---	---
8	---	---	twigs	3652	7	---	---
9	100	27	pellets	2817	56	2346	48
10	150	127	pellets	2220	44	2346	51
11	150	147	pellets	2789	55	2346	53
12	300	274	pellets	3356	67	2346	61
13	300	266	pellets	3048	60	2346	53
14	300	270	pellets	3411	68	2346	65
15	---	---	twigs	3125	62	2346	57

--- No data available

The video of two experiments was annotated using the developed annotation software. In the future, it is planned to expand the base of annotated videos for verification of the software.

3.4 Detector testing results

The average value of the F1 score (formula 6) in the studied videos was 83% for the DoG algorithm and 73% for the LoG algorithm. The analysis showed that both algorithms can provide the necessary accuracy for the detection of firebrands and are comparable in time and accuracy, but the DoG algorithm is easier controlled and implemented.

In the future work, it is planned to more precisely select the main parameters of the LoG and DoG algorithms, namely the variation range of the Gaussian parameter σ and the threshold value of the local brightness maxima in the image.

3.5 Tracker testing results

The results of testing the quality of trackers are given in Table 2. The average values of all tested videos are indicated as the MOTA and MOTP metrics.

Table 2. Evaluating the quality of tracking algorithms.

Algorithm of tracker and detector	MOTP, %	MOTA,%
Based on the near-	42%	31%

est neighbor, LoG		
Based on the nearest neighbor, DoG	48%	51%
Based on the Hungarian algorithm, LoG	41%	49%
Based on the Hungarian algorithm, DoG	49%	62%

The accuracy of metrics ranges from 0 to 100%, where 100% is absolute coincidence.

The analysis showed the superiority of Hungarian algorithm-based trackers. The MOTA and MOTP metrics have a close or higher accuracy for the Hungarian algorithm. At present, the best result is demonstrated by the method based on the DoG tracking algorithm and the Hungarian algorithm (49% and 62%, respectively). The obtained accuracy is in good agreement with the results of other works. For example, multi object tracking studies [34, 35] show comparable accuracy of work using the applied tracker algorithms.

In the future, it is planned to use a Hungarian algorithm-based tracker to improve the accuracy of its work due to additional selecting the parameters of used metrics and improving the quality of detectors. The accuracy of the detectors used is a comparable value.

4 Conclusion

A series of semi-field experiments was conducted on a unique experimental setup that simulated the transfer of burning and glowing firebrands of pine bark and twigs, as well as wood pellets, depending on the number of particles and different recording parameters using high-resolution infrared cameras. A set of videos was obtained, which was later used for testing and verification of the software developed.

To detect firebrands in the thermograms, the Laplacian of Gaussian (LoG) and the Difference of Gaussian (DoG) algorithms were tested. To evaluate the accuracy of the detectors, an original approach applying the F1 score metric was used. The analysis showed that both metrics can provide the necessary accuracy for the detection of firebrands and are comparable in time and accuracy, but the DoG algorithm is easier controlled and implemented.

Different firebrand tracking algorithms have been developed and tested. In particular, a Hungarian algorithm-based tracker (Kuhn-Munkres algorithm) that tracks the movement of detection between frames with a higher accuracy was implemented. The analysis showed that the Hungarian algorithm-based trackers tracked more precisely the movement of firebrands.

A graphical user interface (GUI) was developed for working with the software and creating an annotated video database. The GUI can be used to perform various ma-

nipulations with frames, run a selected detector and a tracker with the specified parameters, as well as to receive the results in the video file.

The further work will be aimed at expanding the database of annotated videos and improving the accuracy of the selected detector and tracker algorithms.

5 Acknowledgements

This work was supported by the Russian Foundation for Basic Research (project #18-07-00548), the Tomsk State University Academic D.I. Mendeleev Fund Program, the Fundamental Research Program of State Academies of Sciences for 2013–2020 (project II.10.3.8) and the Bushfire and Natural Hazard Cooperative Research Centre.

References

1. Mell, W.E., Manzello, S.L., Maranghides, A., Butry, D., Rehm, R.G.: The wildland–urban interface fire problem – current approaches and research needs. International Journal of Wildland Fire **19**, 238–251 (2010). doi: 10.1071/WF07131.
2. Cohen, J.D.: What is the Wildland Fire Threat to Homes? USDA Forest Service Gen. Tech. Rep. PSW-GTR-173. 189–195 (2000).
3. Kornakova, M.; March, A.: Activities in defendable space areas: Reflections on the Wye River-Separation Creek fire. Australian Journal of Emergency Management **32**, 60–66 (2017).
4. Seymat, T.: Deadly wildfires: a devastating year for Portugal. Euronews. (2017).
5. Foote, E I., Manzello, S.L., Liu, J.: Characterizing firebrand exposure during wildland-urban interface fires. In: Proceedings of Fire and Materials 2011 Conference, pp. 479–491. Interscience Communications: San Francisco, CA (2011).
6. Albini, F.A.: Spot fire distance from burning trees- a predictive model. General Technical Report INT-56, USDA Forest Service, Ogden, Utah. (1979).
7. Tarifa, C.S., del Notario, P.P., Moreno, F.G.: On the flight paths and lifetimes of burning particles of wood. In: 10th Symposium (International) on Combustion, 1021–1037. The Combustion Institute (1965).
8. Tse, S.D., Fernandez-Pello, A. and C.: On the flight paths of metal particles and embers generated by power lines in high winds - a potential source of wildland fires. Fire Safety Journal **30**(4), 333–356 (1998).
9. Albini, F.A.: Transport of Firebrands by Line Thermals. Combust. Sci. Technol. **32**(5-6), 277–288 (1983).
10. Manzello, S.L., Park, S., Cleary, T.G.: Investigation on the ability of glowing firebrands deposited within crevices to ignite common building materials. Fire Safety Journal **44**(6), 894–900 (2009). doi:10.1016/j.firesaf.2009.05.001.
11. Suzuki, S., Manzello, S.: Characteristics of Firebrands Collected from Actual Urban Fires. Fire Technology **54**(6), 1533–1546 (2018). https://doi.org/10.1007/s10694-018-0751-x.
12. Filkov, A.; Prohanov, S. Particle Tracking and Detection Software for Firebrands Characterization in Wildland Fires. Fire Technology 1–20 (2018). doi:10.1007/s10694-018-0805-0.

13. Kasymov, D.P., Perminov, V.A., Reyno, V.V., Filkov, A.I., Loboda, E.L.: Experimental setup for producing firebrands to study the spread of wildland fires. Russian Physics Journal **60**(12/2), 107–112 (2017).
14. Loboda, E.L., Kasymov, D.P., Filkov, A.I., Reyno, V.V., Agafontsev, M.V.: Some aspects of field and laboratory research of wildland fires using thermography. In: 30th Intern. Sci.-Pract. Conf. M .: VNIIPO, pp. 295–300 (2018).
15. Kasymov, D.P., Agafontsev, A.M., Filkov, A.I., Perminov, V.V., Reyno, V.V.: Experimental data on the transfer of burning and glowing firebrands and the conditions of ignition of fuel bed. In: the 24th Intern. Symposium, pp. 63–66. Atmospheric and Oceanic Optics. Atmospheric Physics: Proceedings, Tomsk, (2018).
16. Kasymov, D.P., Perminov, V.V., Filkov, A.I., Agafontsev, A.M., Reyno, V.V., Gordeev, E.V.: Generator of burning and glowing firebrands. Patent RF, No. 183063. 2018.
17. El Houssami, M., Mueller, E., Filkov, A. et al.: Experimental procedures characteris-ing firebrand generation in wildland fires. Fire Technology **52**, 731–751 (2016). https://doi.org/ 10.1007/s10694-015-0492-z
18. Filkov, A.I., Prohanov, S.A., Mueller, E., Kasymov, D.P. et al.: Investigation of firebrand production during prescribed fires conducted in a pine forest. Proceedings of the Combustion Institute **36**(2), 3263–3270. (2017).
19. Thomas, J.C., Mueller, E.V., Santamaria, S. et al.: Investigation of firebrand generation from an experimental fire: development of a reliable data collection methodology. Fire Safety Journal **91**, 864–871 (2017). https://doi.org/10.1016/j.firesaf.2017.04.002.
20. Tkinter -Python interface to Tcl/Tk. https://docs.python.org/3/library/tkinter.html, last accessed 2018/12/21.
21. PyGObject. https://pygobject.readthedocs.io, last accessed 2018/12/22.
22. The GUI toolkit for Python. https://wxpython.org/, last accessed 2018/12/22.
23. Python software foundation. https://pypi.org/project/PyQt5/, last accessed 2018/12/23.
24. Tcl Developer Xchange. https://www.tcl.tk/, last accessed 2018/12/23.
25. wxWidgets Cross-Platform GUI Labrary. https://www.wxwidgets.org/, last accessed 2018/01/12.
26. The Qt Company. https://www.qt.io, last accessed 2018/01/12.
27. Qwt User's Guide 6.1.4. https://qwt.sourceforge.io/, last accessed 2018/01/15.
28. Sasaki, Y.: The truth of the F-measure. Teach Tutor mater **1**(5), 1–5 (2007).
29. Knuth, D.: The Art of Computer Programming. Fundamental Algorithms 1(2), (1973).
30. Munkres, J.: Algorithms for the Assignment and Transportation Problems. Journal of the Society for Industrial and Applied Mathematics **5**(1), 32–38 (1957).
31. Bernardin, K., Stiefelhagen, R.: Evaluating multiple object tracking performance: The CLEAR MOT metrics. Eurasip Journal on Image and Video Processing (2008). doi:10.1155/2008/246309.
32. Introducing JSON. https://json.org/, last accessed 2018/01/20.
33. PyInstaller freezes (packages) Python applications into stand-alone executables, under Windows, GNU/Linux, Mac OS X, FreeBSD, Solaris and AIX. http://www.pyinstaller.org, last accessed 2018/01/20.
34. Rahul, M.V., Ambareesh, R., Shobha, G.: Siamese network for underwater multiple object tracking. In: the ACM International Conference, pp. 511–516 (2017).
35. Hua, W., Mu, D., Zheng, Z., Guo, D.: Online multi-person tracking assist by high-performance detection. Journal of Supercomputing, 1–19 (2017).

An editor for teaching the proof of statements for sets

Vadim Rublev[1] and Vladislav Bondarenko[2]

[1] Department of Theoretical Computer Science, Demidov Yaroslavl State University, Yaroslavl, 150000, Russia,
roublev@mail.ru,
ORCID 0000-0002-0252- 9958
[2] Department of Theoretical Computer Science, Demidov Yaroslavl State University, Yaroslavl, 150000, Russia,
bondarencko40@gmail.com

Abstract. The construction of an editor for teaching the proof of statements for sets, which should become the main part of the computer teaching system for proving statements for sets, is considered. The problems of the editor organization, allowing a step by step proof and control of each step correctness, are solved.

Keywords: computer training, proving statements for sets, editor for teaching the proof, step by step proof, control of step correctness

1 Introduction

At present, the problems of teaching the bulk of students in mathematical and computer Sciences are associated with underdevelopment of thinking caused by the low quality of school education. A significant mass of school graduates do not know how to read (understand what they read), do not know how to reason, because they were not taught enough. These problems can be solved only by individual training, but this approach requires a huge additional time from the teacher, exceeding several times the hours allocated by the curriculum. Therefore, it suggests a solution in the development of computer automated learning systems (ALS), with which you can not only control the knowledge, but to conduct training. Basically, many computer systems control the testing of student memory, and therefore can teach some definitions, but not their use. ASL for the exact Sciences should teach data analysis, formalization, analysis, reasoning and transformation. The use of computer algebra [1] underlies the construction of such systems. For example, one of the authors of this paper used this to construct ASL of computational complexity of algorithms[2].

In this paper, models of computer-based learning to prove statements for sets are considered. These models can be divided into 2 groups: proof-of-statement models for sets and training models that prepare a student for the first group models. The models of the first group are described in the proof of statements editor for sets and the study of models of the second group related to the training of using models of the first group is supposed to be.

2 The problem of constructing a proof editor of statements for sets

To solve the problem specified in the headline, select the following task sequence:

1. Limitations on the type of statements for sets for whose prove you need to build an editor.
2. Equivalent transformation of the sets included in the statement.
3. Splitting the basic statement into an equivalent set of simple statements.
4. The choice of the method of proving a simple statement and the selection of an initial set of premises in it.
5. Definition of the elementary step of the proof.
6. Control of the correctness proof by the editor.

2.1 Statement type constraints for sets

In the general case of the statement we will consider some *universal* set U and its subsets $X_1, X_2, \ldots, X_n$. The statement uses formulas for these subsets, constructed with operations *complement, intersection, union*, and brackets, changing the order of these operations, if necessary.

We restrict ourselves to statements for sets of the following form

$<$ *Constriction of the set relation* $>$ $\{\rightarrow \mid \leftrightarrow\}$ $<$ *Constriction of the set relation* $>$

So in example (1)

$$\overline{X_1 \cap \overline{X}_2 \cup \overline{X}_3} = (X_2 \cup X_3) \cap \overline{X}_1 \;\leftrightarrow\; X_1 \cap X_2 \cap X_3 = \emptyset \;\wedge\; X_2 \subseteq X_1 \cup X_3 \quad (1)$$

it is argued that the equality of two sets given by expressions takes place if and only if the intersection of the subsets is empty and the second subset is included in the union of the other two.

2.2 Equivalent transformation of sets included in the statement

To simplify the process of a proof conducted by a student, it is recommended to simplify the complex expressions of some sets of statements. In this case, we mean the representation of a complex expression in the form of a union of sets (or their intersections) in some cases or as the intersection of sets (and their unions) in other cases. So in example (1) transformation of the set on the left side of the set equality (it by A) denote

$$A \equiv \overline{X_1 \cap \overline{X}_2 \cup \overline{X}_3} = (\overline{X}_1 \cup X_2) \cap X_3 = \overline{X}_1 \cap X_3 \cup X_2 \cup X_3 \quad (2)$$

gives both views (2). The set of the right part of the equation (it by B) denote is already represented in (1) by an intersection, and the union is obtained by the following transformation

$$B \equiv (X_2 \cup X_3) \cap \overline{X}_1 = X_2 \cap \overline{X}_1 \cup X_3 \cap \overline{X}_1. \quad (3)$$

The reason why such representations of the set are important is the possibilities of simplifying the conduct of the proof. The representation of a set as a union of its subsets allows us to divide the further proof into separate branches, where the membership of an element to a set becomes easier by dividing it into cases of belonging to a particular subset, and the proof for each such branch is simplified and can be conducted separately. The representation in the form of an intersection simplifies the proof of the conclusion that the element does not belong to the intersection of sets, because it is sufficient to obtain a result about its non-belonging to one of the sets belonging to the intersection.

2.3 Splitting the basic statement into an equivalent set of simple statements

If there is an equivalent operation in the basic statement, the statement can be replaced by a conjunction of two statements with implications in one direction and the other. So the statement example (1) can be replaced by the conjunction of the following statements (with the replacement of the parts of the set equality by the introduced notations A and B):

$$A = B \ \rightarrow \ X_1 \cap X_2 \cap X_3 = \emptyset \ \wedge \ X_2 \subseteq X_1 \cup X_3 \tag{4}$$

$$X_1 \cap X_2 \cap X_3 = \emptyset \ \wedge \ X_2 \subseteq X_1 \cup X_3 \ \rightarrow \ A = B \tag{5}$$

Note that statement (4), having in conclusion the conjunction, can be spitted into 2 statements. In statement (5), the relation of equality of 2 sets at the end of the implication can be replaced by the conjunction of 2 inclusion, and therefore the statement (5) can also be divided into 2. As a result, we obtain the following partition of the basic statement (1):

$$A = B \ \rightarrow \ X_1 \cap X_2 \cap X_3 = \emptyset \tag{6}$$

$$A = B \ \rightarrow \ X_2 \subseteq X_1 \cup X_3 \tag{7}$$

$$X_1 \cap X_2 \cap X_3 = \emptyset \ \wedge \ X_2 \subseteq X_1 \cup X_3 \ \rightarrow \ A \ \subseteq \ B \tag{8}$$

$$X_1 \cap X_2 \cap X_3 = \emptyset \ \wedge \ X_2 \subseteq X_1 \cup X_3 \ \rightarrow \ B \ \subseteq \ A \tag{9}$$

An example of reducing the proof of the basic statement (1) to the proof of 4 simple statements (6)-(9) shows that this can be done in other cases of the basic statements.

2.4 The choice of a method of proving a simple statement and the selection of the initial set of premises in it

In a simple statement, when performing the premises of the left part of the implication, it is necessary to prove the truth of the right part of the implication(call it a *target* statement). There are 2 methods of proof when the target is the inclusion relation of sets:

– **direct** method when we prove for an *arbitrary* element of the universal set, that from the belonging of an element to the included set follows its belonging to the including set (call it a *target conclusion*), that is, the accuracy of the inclusion.
– **indirect** method, when we assume the opposite in the target statement(*exists an element of the universal set that belongs to the included set of the target relation of the inclusion of sets, but does not belong to the including set*) and by a consistent conclusion come to a contradiction with the conjunction of premises left part of the implication to this simple statement.

For example, in simple statements (7)-(9) both methods are possible. However, for statement (7) it is more rational to apply the method (perhaps indirect fewer steps of proof), and for statements (8)and (9) the direct method is more rational.

If the target statement is the equality of a set to an empty set or to a universal set, then only the indirect method is rational. In this case, the existence of its element is opposite for an empty set, and the existence of an element that does not belong to this set is opposite for the equality of a certain set to a universal set. For a simple statement (6) you need to use an indirect method and show the contradiction with the premises of the statement.

The proof of any method begins with sequential steps, each of which is based on a premise. The initial premise in the direct method of proof is that an arbitrary element of the universal set belongs to the included set of the target statement. For example, for statement (8), the initial premise
is the expression $\forall x : x \in A$.

The initial premise in the indirect method of proof is the denial of the target statement, which is expressed by the existence of an element that contradicts the target statement. For example, for (6), the initial statement premise is the expression $\exists x : x \in X_1 \cap X_2 \cap X_3$, and for (7) – statement $\exists x : x \in X_2, x \notin X_1 \cup X_3$.

In addition to the initial premise, conclusions can be based on assumptions statement related to the conditions (left side of the implication).The statement system prepares them as the initial set of premises, adding to it the initial premise. Meanwhile

1) to the equality of sets (for example, $C = D$) there correspond 4 premises (in the example, $x \in C \to x \in D$, $x \in D \to x \in C$, $x \notin C \to x \notin D$, $x \notin D \to x \notin C$);
2) to the inclusion case (for example, $X_2 \subseteq X_1 \cup X_3$) there correspond 2 premises (in the example, $x \in X_2 \to x \in X_1 \cup X_3$ and $x \notin X_1 \cup X_3 \to x \notin X_2$);
3) to the equality of a set to the empty set (for example, $X_1 \cap X_2 \cap X_3 = \emptyset$) there correspond 1 premise (in the example, $x \notin X_1 \cap X_2 \cap X_3$);
4) and the equality of a set to the universal set (for example, $X_1 \cup X_2 \cup X_3 = U$) also matches 1 premise (in the example, $x \in X_1 \cup X_2 \cup X_3$).

For each elementary conclusion of the proof, if it is true, the system adds the conclusion as a premise to the set of premises of the proof or branch of the proof (more on that in the next section).

2.5 Definition of the elementary step of the proof

The proof is conducted with the help of a sequence of steps, at each of which an elementary conclusion is drawn, based on the indicated premises for conclusion. For educational purposes, we limit ourselves to only elementary conclusions based on no more than 2 premises. For elementary conclusions, the following ideas are used:

1. If an element belongs to a certain set (for example, premises $x \in C$), the conclusion can be its belonging to any set, covering the set of the premises (in the example, the conclusion $x \in C \cup D$).
2. If an element belongs to two sets (for example, 2 premises $x \in C$ and $x \in D$), the conclusion can be an element belonging to the intersection of these sets (in the example conclusion $x \in C \cap D$).
3. If an element belongs to a set (for example , $x \in C \cap D$), it belongs to each part of it (in the example , 2 conclusions $x \in C$ and $x \in D$).
4. If an element belongs to a set (for example, the premise $x \in C$), it does not belong to its complement (in the example the conclusion $x \notin \overline{C}$).
5. If an element does not belong to a set (for example, premise $x \notin C$), it belongs to its complement (in the example, the conclusion $x \in \overline{C}$)).
6. If an element does not belong to two sets (for example, 2 premises $x \notin C$ and $x \notin D$), it does not belong to the union of these sets (in the example the conclusion $x \notin C \cup D$).
7. If an element does not belong to a union of sets (for example, the premise $x \notin C \cup D$), it does not belong to any of these union sets (in the example, the conclusion $x \notin C$ and the conclusion $x \notin D$).
8. If an element belongs to a union of sets (for example, the premise $x \in C \cup D$), it can belong to one of these union sets (in the example, the conclusion $x \in C$ and the conclusion $x \in D$) – this conclusion is called *the splitting of the cases.*

Note that the splitting into cases can be conducted in different ways. A partition where the sets of cases do not intersect is called *alternative.* As an example of the premise $x \in C \cup D$ you can write the following division into cases: the conclusion $x \in C$ and the conclusion $x \in D \cap \overline{C}$. This is especially important when for one case, the further conclusion is easily built. Then, for the second case, additional information is obtained, which can help in the further conclusion. If the first case is also difficult, it can be divided into 3 cases: the conclusion $x \in C \cap \overline{D}$, the conclusion $x \in C \cap D$ and the conclusion $x \in D \cap \overline{C}$.

Note also that the division into alternative 2 cases of belonging to a certain or set non-belonging to this set can always be done without relying on premises (for example, the conclusion $x \in C$ and the conclusion $x \notin \overline{C}$ form 2 cases and do not require a premise).

Each conclusion, if made correctly, is added as a premise to the preceding set of premises. But when cases appear, each of them is associated with its own independent branch of proof and many premises of this branch, which is formed from the previous set of premises by adding a new premise – the case conclusion.

Each branch of the proof must end with either a white square denoting the receipt of the target conclusion, or a black square denoting the receipt of a contradiction. If all branches of the proof from the opposite ended with a contradiction, the proof of the statement was successful. If in the direct method the proof of all branches ended, but there are branches that ended in success (a white square), the proof was successful.

2.6 Editor control of the proof

The system allows the trainee to build a proof of the basic statement. But the system has to control all his or her actions, starting from splitting into simple statements, selecting the method of proof with the organization of an initial premise, performing each elementary step of the proof up to the completion of each branch of the proof.

To this purpose, the system, at the end of the above actions (by pressing a button **Verify**), builds a Boolean function corresponding to a statement for sets (simple statements into which the basic statement is splitted, or statements of the original premise, or the statement of an elementary step, as implications of conjunctions of the premises and conclusion of the elementary step or implication of the premise and disjunction of the conclusion of cases) and verifies its identity truth (truth on any argument sets). This Boolean function (let us call it *BIF* Boolean Identification Function) is constructed as follows:

1. Each set X_i is replaced by a boolean variable y_i, whose value is equal to the truth of the statement that the element x belongs to this set ($y_i \equiv (x \in X_i)$). Other are replaced in the same way. For example, the set A is replaced by a boolean variable a, whose name is a lowercase letter, corresponding to the set name and its true value coincides with the value of the statement of the belonging of an element x to this set, i.e. $a \equiv (x \in A)$.
2. Operations on sets of complement, intersection and union are replaced accordingly by operations of negation, conjunction and disjunction for the corresponding subsets of statements.
3. The equality for sets is replaced by the equivalence operation of the corresponding statements.
4. The inclusion for sets is replaced by the operation of implication of the corresponding statements.
5. The equality of a set to the empty set is replaced by the negation of the corresponding statement for the set.
6. The equality of a set to the universal set is replaced by the corresponding statement for the set.

So in the considered example (2) of identical representations of the set A we obtain the following BIF $f_a = (\overline{y_1 \wedge \overline{y_2} \vee \overline{y_3}} \leftrightarrow (\overline{y_1} \vee y_2) \wedge y_3) \wedge (\overline{y_1 \wedge \overline{y_2} \vee \overline{y_3}} \leftrightarrow (\overline{y_1} \wedge y_3) \vee (y_2 \wedge y_3))$ and since it is identically true, then by the theorem *on the connection of expressions of the algebra of sets and the algebra of statements* and its consequences (see, for example [5]) the system confirms the correctness of transformations of the set A.

Next, for a simple statement (6), the BIF will look as follows:

$$f_6 \equiv (\overline{y_1 \wedge \overline{y_2} \vee \overline{y_3}} \leftrightarrow (y_2 \vee y_3) \wedge \overline{y_1}) \rightarrow \overline{y_1 \wedge y_2 \wedge y_3}.$$

Its identical truth also confirms the correctness of the simple statement (6).

When conducting the proof from the contrary of this statement, the student receives the initial premise. $\exists x : x \in X_1 \cap X_2 \cap X_3$ from the negation of the proved statement $\overline{X_1 \cap X_2 \cap X_3 = \emptyset}$ and BIF for verification gets the following expression: $f_{u1} \equiv y_1 \wedge y_2 \wedge y_3 \leftrightarrow \overline{\overline{y_1 \wedge y_2 \wedge y_3}}$, and since it is identically true, the initial premise is correctly built by the student.

From the received initial premise follows a conjunction of 3 elementary conclusions. $x \in X_1 \wedge x \in X_2 \wedge x \in X_5$, what is verified by the identical truth of BIF of the function of implication of the function of premise and conjunction of functions of elementary conclusions $y_1 \wedge y_2 \wedge y_3 \rightarrow y_1 \wedge y_2 \wedge y_3$.

3 Editor interface for proving assertions for sets

The editor interface is a constructor. Using a dialog with a choice of actions (Fig. 1), the trainee chooses which elements he will need for further work. Such elements will appear in the main program window (Fig. 2). Another single window is the input toolbar (Fig. 3). The proof process is divided into several stages:

1. Initially, at launch, a dialog is called up with the choice of action (Fig. 1), where learner selects one of the items he needs for the proof.
2. “Enter the basic statement” creates a field for input.
3. “Inputting identical transformations for some sets” (they are given the names A and B create two input fields with buttons for checking the transformations).
4. “Splitting the basic statement” creates a split number, an input field and a validation button.
5. “The choice of the statement to be proved” creates a dialog window in which the previously entered splits of the basic statement are displayed.
6. “Entering the initial premise and step of proving” creates verify button and input field.

After entering the text of the elementary conclusion or the initial premise into the scheme of the proof, one or several buttons are displayed in the form of a red square, pressing one of them will form a field for entering the text of the next elementary conclusion. The choice of an elementary conclusion of a white or black square, if correct, leads to the completion of the proof of the statement or its branch.

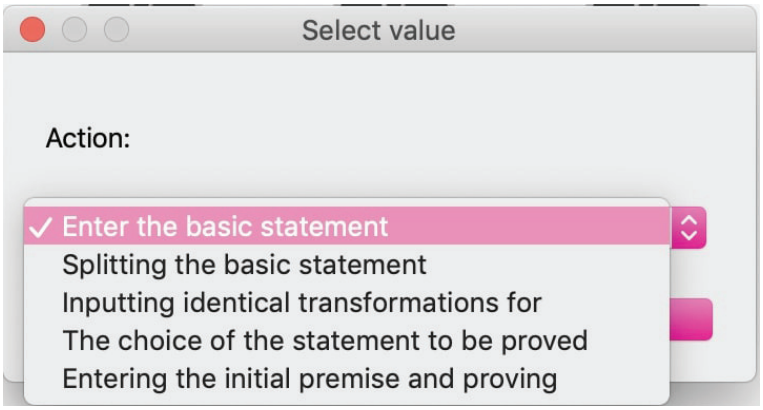


Fig. 1: Choose dialog

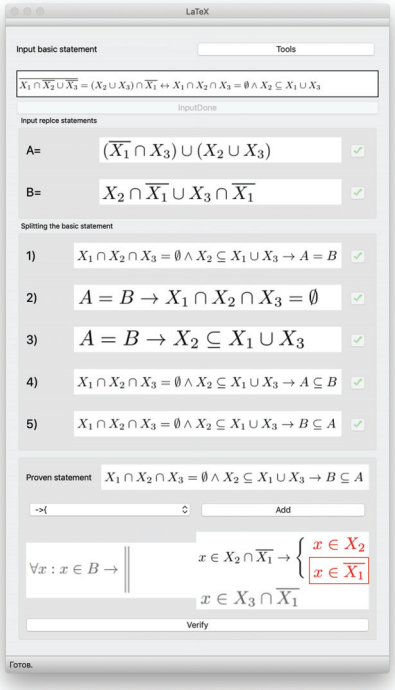


Fig. 2: A window for entering the required statements and premises of evidence

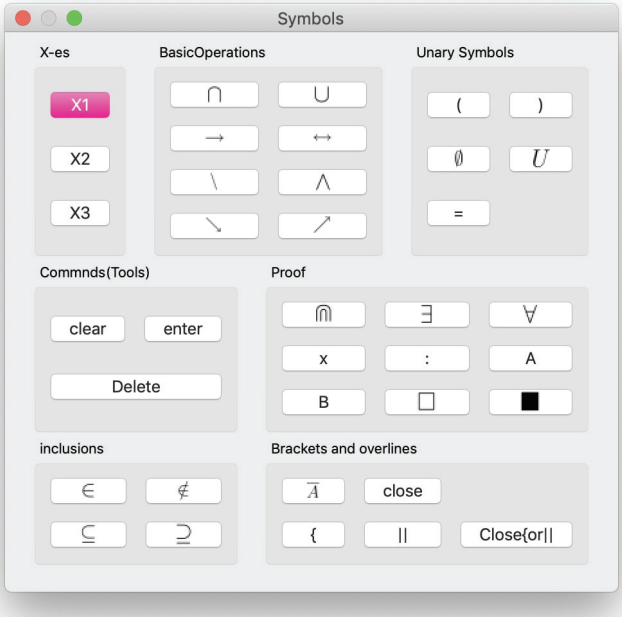


Fig. 3: Window of tools required for proof

4 Conclusion

All the tasks of developing an editor for proving statements for sets are achieved, and we hope that this will allow us to complete the development of the ALS, where this editor will be the central construction teaching this material.

References

1. Davenport J.H., Siret Y., Tournier E.: Computer algebra: systems and algorithms for algebraic computation / - Academic Press, 1988. ISBN 978-0-12-204230-0
2. Rublev V.S., Yusufov M.T.: Automated system for teaching computational complexity of algoritm cours, Convergent Cognitive Information Technologies (Selected Papers of the First International Sientific Conference Convergent Cognitive) Moscow, Russia, November 25-26, 2016 (http://ceur-ws.org/Vol-1763/). (ISSN 1613-0073 VOL-1763 urn.nbn.de: 0074-1763-4)

Nobrainer: an Example-driven Framework for C/C++ Code Transformations

Valeriy Savchenko[1], Konstantin Sorokin[1], Georgiy Pankratenko[1], Sergey Markov[1], Alexander Spiridonov[1], Ilia Alexandrov[1], and Alexander Volkov[2]

[1] Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russian Federation
{vsavchenko,ksorokin,gpankratenko,markov,aspiridonov,ialexandrov}@ispras.ru
[2] Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation
asvolkov@ispras.ru

Abstract. Refactoring is a standard part of any modern development cycle. It helps to reduce technical debt and keep software projects healthy. However, in many cases refactoring requires that transformations are applied globally across multiple files. Applying them manually involves large amounts of monotonous work. Nevertheless, automatic tools are severely underused because users find them unreliable, difficult to adopt, and not customizable enough.

This paper presents a new code transformation framework. It delivers an intuitive way to specify the expected outcome of a transformation applied within the whole project. The user provides simple C/C++ code snippets that serve as examples of what the code should look like before and after the transformation. Due to the absence of any additional abstractions (such as domain-specific languages), we believe this approach flattens the learning curve, making adoption easier.

Besides using the source code of the provided snippets, the framework also operates at the AST level. This gives it a deeper understanding of the program, which allows it to validate the correctness of the transformation and match the exact cases required by the user.

Keywords: code transformation · global refactoring · C/C++.

1 Introduction

The lifecycle of any software project is a constant evolution. Not only does it mean writing new code while adding new features, but it also includes continuously modifying the existing code. Excessive focus on extending the system's functionality can lead to a rapid accumulation of the project's technical debt. The concept of *technical debt* is a widespread metaphor for design-wise imperfection that boosts initial product development and deployment. With time, however, the debt grows larger and can potentially stall the whole organization [3].

A common way to mitigate this problem is *refactoring* [2, 12], which is a modification of the system's internal structure that does not change its external

behavior [4]. It helps to eliminate existing architectural flaws and ease further code maintenance. Murphy-Hill et al. [9] have estimated that 41% of all programming activities involve refactoring. The same study also concluded that developers underuse automatic tools and perform code transformations manually despite the fact that a manual approach is more error-prone. Research performed on StackOverflow website data [10] found that corresponding tools can be too difficult and unreliable, as well as require too much human interaction. This reveals a few natural requirements for a beneficial refactoring tool — it should be easy to use, ask a minimal number of questions from the end user, and rely on syntactic and semantic information in order to ensure the correctness of the performed transformations.

Highly customizable refactoring tools typically utilize additional domain-specific languages (DSL) for describing user-defined transformation rules [5,7,14]. Such languages need to express both the intended refactoring and the different syntactical and semantical structures of the target programming language. Adopting a DSL can be too overwhelming in the case of C/C++ languages because the language itself is already complex. Studies show that C and C++ take longer to learn [8], and projects in these languages are more error-prone [11] compared to other popular languages.

This insight further qualifies the ease-of-use requirement: the tool should not introduce another level of sophistication on top of C/C++ nor expect additional knowledge from its user.

This study presents the Clang-based transformation framework `nobrainer`, which is built on such principles. The expression *a no-brainer* refers to something so simple or obvious that you do not need to think much about it.[3][4] This concept reflects the core idea behind `nobrainer`, the idea of providing an easy-to-use framework for implementing and applying a user's own code transformations.

Individual `nobrainer` rules are written in C/C++ without any DSLs. Each rule is a group of examples that represents situations that should be refactored and illustrates the way they should be refactored. They look exactly like developer's code, thus flattening the learning curve of the instrument.

In this paper, we describe the main principles behind `nobrainer`, illustrate the most interesting design and implementation solutions, and demonstrate the tool's application in real-world scenarios.

2 Related work

This section covers the various approaches on code transformation and automatic refactoring presented in the literature. We distinguish two key points in the current review. The first point is the form, in which the transformations are described. The second point is the way these transformations are performed. There are a few similar approaches that can be combined and compared.

[3] `https://www.merriam-webster.com/dictionary/no-brainer`

[4] `https://dictionary.cambridge.org/dictionary/english/no-brainer`

Most of the tools reviewed here rely on a separate syntax for describing transformation or refactoring rules. For example, Waddington el al. [5] introduce their language YATL; whereas, Lahoda et al. [7] extend the Java language to simplify rule descriptions. We believe that various types of DSLs can confuse the user and introduce another layer of complexity. Wright and Jasper [14] describe a different approach. Their tool `ClangMR` adopts Clang AST matchers [1] as a mechanism for describing the parts of the user's code that should be transformed. The user must define replacements in terms of AST nodes. The authors imply that the user is familiar with the principles of syntax trees and how it is built for C/C++ programming languages. We believe that this requirement is rarely met, and that is why the adoption of `ClangMR` can be challenging for a regular user.

Wasserman [13], on the other hand, introduces a tool (`Refaster`) that does not involve any DSLs. He suggests using the target project's programming language for describing transformations. This allows the user to embed transformation rules into the project's code base, which leads to simpler syntax checks and symbol availability validations. Transformation rules are written in the form of classes and methods with either `@BeforeTemplate` or `@AfterTemplate` annotations. Each class represents a transformation and should contain one or more `@BeforeTemplate` methods and a single `@AfterTemplate` method. Then the tool treats the transformation as follows: match the code that is written in `@BeforeTemplate` method and replace it with the code written in `@AfterTemplate` method.

We consider Wasserman's tool design to be clear and user-friendly because it uses the language of the project's code base to define transformations. We use a similar approach in `nobrainer`.

We decided that the best method for matching the C/C++ source code is the approach used in `ClangMR`. However, because using Clang AST matchers directly can be challenging, we provide a higher level framework that utilizes AST matchers internally.

Regarding the code transformation, a common solution is to generate an AST, transform it, and restore the source code in the end. This kind of approach is used by `Proteus` [5], `Jackpot` [7] and Eclipse C++ Tooling Plugin [6]. The main problem of implementing such an approach is code generation. We should remember all the nuances of the original source code in order to replicate them when restoring the resulting code. This includes preserving comments, redundant spaces, etc. On the other hand, in `ClangMR` [14], the authors suggest using the `Clang`[5] framework for code transformation because it allows developers to directly modify the source code token wise. We also use the Clang framework because we believe it is the best solution to transform C/C++ source code.

[5] https://clang.llvm.org/

3 Design

In this section, we describe the overall design and the user's workflow. Running the tool on a real project involves the following list of actions:

- writing transformation rules as part of the target project
- providing compilation commands for the target project (the currently supported format is the Clang compilation database[6])

`Nobrainer` either applies all the replacements or generates a YAML file containing these replacements. In the latter case, replacements can be applied later with the `clang-apply-replacements` tool (part of the Clang Extra Tools[7]).

Fig. 1 provides an insight into the internal `nobrainer` structure. Each numbered block represents a work phase of the tool. Boxes at the bottom correspond to each phase's output.

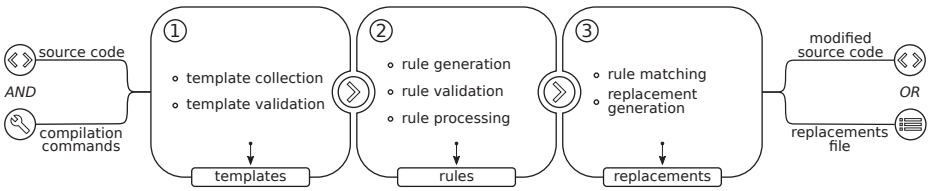


Fig. 1. Nobrainer workflow

During the first phase, the tool analyzes all of the translation units that are extracted from the given compilation commands. For each translation unit, it searches for and collects templates that represent our transformation examples. Then `nobrainer` filters invalid templates. The result of the first phase is a list of valid templates.

In the second phase, we group conforming templates into rules. `Nobrainer` also checks each rule for consistency and then processes each rule to generate internal template representation.

In the third phase, we work with the list of preprocessed rules. `Nobrainer` tries to match each rule against the project's source code. For each match, we construct a special data structure, which we use to generate a replacement. As a result, we obtain a set of replacements.

For more details on each phase, see Section 4.

4 Detailed description

The core idea of `nobrainer` is the use of `examples`, which are code snippets written in C/C++ languages. Because each snippet may represent a whole fam-

[6]https://clang.llvm.org/docs/JSONCompilationDatabase.html
[7]https://clang.llvm.org/extra/index.html

ily of cases, we call them *templates*. The user submits the situations she wants to change in the `Before` templates and the substituting code in the `After` templates. These templates can be defined anywhere in the project.

`Nobrainer` offers a special API for writing such examples, which is subdivided into C and C++ APIs. Both provide the ability to write `expression` and `statement` templates to match C/C++ expressions or statements respectively.

For a clearer explanation of what a template is, let us proceed with an example. Suppose the user wants to find all calls to function `foo` with an arbitrary `int` expression as the first argument and global variable `globalVar` as the second argument and replace the function with `bar`, while keeping all the same arguments. Listing 1 demonstrates how such a rule can be specified (using `nobrainer` C API).

```
int NOB_BEFORE_EXPR(ruleId)(int a) {
  return foo(a, globalVar);
}

int NOB_AFTER_EXPR(ruleId)(int a) {
  return bar(a, globalVar);
}
```

Listing 1: Expression template example

Expressions for matching and substitution reside inside of `return` statements. We force this limitation intentionally because it allows us to delegate the type compatibility check of `Before` and `After` expressions to the compiler.

`Nobrainer`'s transformations are based on the concept that two valid expressions of the same type are syntactically interchangeable. This statement is correct with the exception of parenthesis placement. In certain contexts, some expressions must be surrounded with parentheses. However, we introduce a simple set of rules that solve this issue and are not covered in this paper.

In order to properly define the term *template*, we first need to introduce the following notations (with respect to the given program):

- Θ is a finite set of all types defined
- Σ is a finite set of all defined symbols (functions, variables, types)
- $\mathcal{A}$ is a finite set of all AST nodes representing the program
- $\mathcal{C}$ is a finite set of characters allowed for C/C++ identifiers
- $\mathcal{P}$ is a finite set of all function parameters $p = \langle n_p, t_p \rangle$ where $n_p \in \mathcal{C}^*$ is the parameter's name and $t_p \in \Theta$ is its type

An expression template can be formally defined as a 6-tuple

$$T_{expr} = \langle k, n, r, P, B, S \rangle \tag{1}$$

where

– $k \in \{\texttt{before}, \texttt{after}\}$ is the template's kind
– $n \in \mathcal{C}^*$ is the rule's identifier, it is used for pairing corresponding `before`/`after` templates
– $r \in \Theta$ is the return type
– $B \subset \mathcal{A}$ is the body
– $P \subseteq \mathcal{P}$ is the set of parameters
– $S \subseteq \Sigma$ is the set of symbols used in B

The last two elements of the tuple require additional commentary.

Template parameters P represent generic placeholders for different situations encountered in the real source code. `Nobrainer` reads these parameters as arbitrary expressions of the corresponding type. For example, parameter `a` from Listing 1 corresponds to **any** expression of type `int`.

The set of symbols S is important for the correctness affirmation (see Sections 4.4 and 4.6).

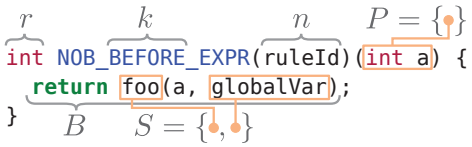


Fig. 2. `Before` template deconstruction

Fig. 2 dissects the `Before` template from Listing 1.

The following subsections cover `nobrainer`'s phases in more detail.

4.1 Template collection

The first phase is to collect all the templates from the project. `Nobrainer` scans each file and tries to find functions that were declared using the API. This can only be done for parsed source files. Doing so for the whole project can have a drastic impact on the tool's performance. In order to avoid checking all the files, we only process files that contain inclusion directives of `nobrainer` API header files.

As the output, this phase has a set of all templates defined by the user. We denote it as $\mathcal{T}$.

4.2 Template validation

After the template collection phase, we validate each template individually. We need to check that the collected templates in $\mathcal{T}$ are structurally valid. First it is important to note that the syntactic correctness of a template is guaranteed by the compilation process. Templates are implemented as the part of the existing code base, which implies that they are actually parsed and checked during the

collection phase. This includes checks for the availability of all symbols, type checks, etc.

In every separate case, `nobrainer` replaces a single expression with another single expression. Considering this fact, each template T_{expr} should define *exactly one* expression. This requirement is transformed into a syntax form as: a template's body B should consist of a single return statement with a non-empty return expression. During the template validation stage, we check this constraint. Thus, `nobrainer` filters out templates without a body (i.e. declarations), templates with an empty body, and templates with a single statement `return;`.

Currently there are some limitations regarding the usage of functional style macros and the usage of C++ lambda expressions in template bodies. For this reason, we validate the absence of either of these language features.

Thus, if `nobrainer` encounters invalid templates, it filters them out and proceeds to the next phase with the set of valid ones $\mathcal{T}_+$.

4.3 Rule generation

For an arbitrary $id \in \mathcal{C}^*$, we define two sets of templates B_{id} and A_{id} as follows:

$$B_{id} = \{T \in \mathcal{T}_+ | n_T = id, k_T = \texttt{before}\} \tag{2}$$

$$A_{id} = \{T \in \mathcal{T}_+ | n_T = id, k_T = \texttt{after}\} \tag{3}$$

These two groups describe exactly one user-defined transformation scenario because they include all of the *Before* and *After* examples under the same name. However, in order for B_{id} and A_{id} to form a transformation rule, the following additional conditions must be met:

$$\begin{cases} |B_{id}| \geq 1 \\ A_{id} = \{a_{id}\} \text{ (i.e. } |A_{id}| = 1) \\ \forall b \in B_{id} \rightarrow a_{id} \prec b \end{cases} \tag{4}$$

where

$$\forall x, y \in \mathcal{T}_+ \rightarrow x \prec y \Leftrightarrow \begin{cases} P_x \subseteq P_y \\ r_x = r_y \end{cases} \tag{5}$$

We refer to operator $\prec$ as the *compatibility operator*. It indicates that the snippet defined in x can safely replace the code matching y. The equality of return types r ensures that the substituting expression has the same type as the original one, while condition $P_x \subseteq P_y$ guarantees that `nobrainer` will have enough expressions to fill all of the x's placeholders.

As a result, we define *transformation rule* as a pair $R_{id} = \langle B_{id}, A_{id} \rangle$ where B_{id} and A_{id} meet conditions (4). Additionally we denote the set of all project rules as $\mathcal{R}$.

4.4 Rule processing

Before template processing As mentioned before, we convert `Before` templates to Clang AST matchers. These provide a convenient way to search for sub-trees that fit the given conditions. They describe each node, its properties, and the properties of its children. Thus, this structure resembles the structure of the AST itself. In order to generate matchers programmatically, we exploit this fact. Each node of the template's sub-tree is recursively traversed and paired with a matcher. As a result, we encapsulate the logic related to different AST nodes and avoid the necessity of supporting an exponential number of possible node combinations.

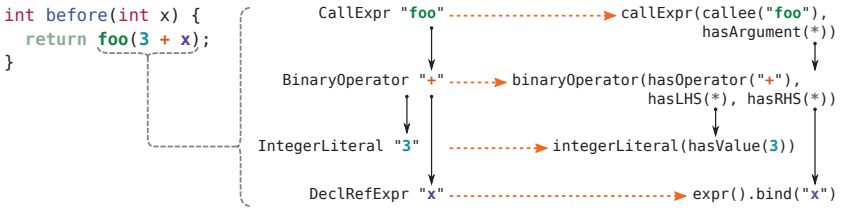


Fig. 3. Recursive AST matcher generation

Fig. 3 demonstrates a simplified example of such a conversion. It depicts three stages of `Before` template processing: source code, AST, and AST matchers. Bold arrows reflect parent-child relationships, while dashed arrows stand for node-matcher correspondence. Because matchers are represented by a series of nested function calls, we construct the innermost matchers first, traversing the tree in a depth-first fashion.

Matching identical sub-trees Consider the `Before` template from Listing 2. It is unlikely that the user expects the system to match two arbitrary expressions as `foo`'s arguments. In fact, the most intuitive interpretation of this template is matching calls to function `foo` with identical arguments only.

```
int before(int x) {
  return foo(x, x);
}
```

Listing 2: An example of reusing a template parameter

Clang does not provide a matcher that can do the job. However, `nobrainer` already has a mechanism to find identical sub-trees for `Before` templates without

parameters. During the matching process, we reuse this mechanism to dynamically generate a matcher. Thus, for the given example, we bind the first argument to `x`, generate a matcher, and check if the second argument fits.

After template processing Our goal is to construct a text that represents the result of a replacement. Therefore, we convert *After* templates into plain strings. However, there are some parts of the *After* template's body that cannot be taken as is and placed into the desired location. We call such parts *mutable*. During the traversal of the *After* template's body, we extract the ranges that represent mutable parts. Each range consists of the start and the end locations of the certain AST node. There are two cases of *mutable* parts.

The first case is the use of a template parameter inside of an *After* template's body. We treat each template parameter as a placeholder that we fill during replacement generation (see Section 4.6).

The second case is the use of a symbol. Inserting symbols in arbitrary places in the source code can be syntactically incorrect. Indeed, in the location of insertion, the symbol may not yet have been declared. Thus, we collect symbol information that is used during replacement generation (see Section 4.6).

Given these points, for the *After* template from Listing 3, we construct the following format string: `"#{bar}(${x}) + 42"`. In this example, `nobrainer` distinguishes the symbol `bar` and the template parameter `x`, and handles them accordingly. The tool treats all the remaining parts of the string as immutable, and, with this in mind, constructs the resulting format string.

```
int after(int x) {
  return bar(x) + 42;
}
```

Listing 3: An example of an *After* template

4.5 Rule application

The next step is to identify all situations, in which to apply rules $\mathcal{R}$. In order to do this across the whole project, `nobrainer` independently parses all the source files. After that, the tool applies AST matchers generated for each rule.

Each time a match is found, `nobrainer` obtains a top-level expression that should be replaced and a list of AST sub-trees bound to parameters from the corresponding *Before* template. Using this information and the *After* template, `nobrainer` generates an actual code change called a *replacement*.

4.6 Replacement generation

Replacement is a sufficient specification for a complete textual transformation. It consists of four components:

– the file where current replacement is applied
– the byte offset where the replaced text starts
– the length of the replaced text
– the replacement text

`Nobrainer` extracts the first three components from the expression marked for substitution. The replacing text is composed from the *After* template and AST nodes bound to parameters. For each node, `nobrainer` gathers the corresponding source code and fills placeholders from the *After* template. This operation results in plain text for the substitution. Fig. 4 demonstrates this procedure using a real code snippet.

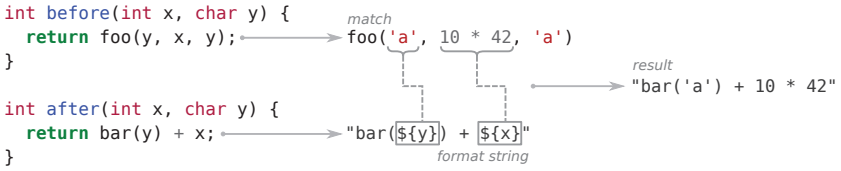


Fig. 4. Replacement generation

Such a transformation may nevertheless cause compilation errors due to symbol availability. `Nobrainer` should check that each symbol that comes with a substitution is declared and has all required name qualifiers. In order to ensure this, we:

– add inclusion directives for the corresponding header files
– add namespace specifiers

The resulting code incorporates only the pieces of real source code that have been checked by Clang at different stages of the analysis.

4.7 Type parameters

Parametrization with arbitrary expressions provides a flexible instrument for generic rule definition. However, this may not be enough. Exact type specification can significantly limit the rule's expressiveness and reduce the number of potential use cases.

In order to combat this shortcoming, we introduce a set of type parameters $\Phi \subset \mathcal{C}^*$ to a template syntax. This extends the template definition (1) to

$$T_{expr} = \langle k, n, r, P, B, S, \Phi \rangle \tag{6}$$

and compatibility operator $\prec$ (5) to

$$\forall x, y \in \mathcal{T}_+ \to x \prec y \Leftrightarrow \begin{cases} \Phi_x \subseteq \Phi_y \\ P_x \subseteq P_y \\ r_x = r_y \end{cases} \tag{7}$$

Note that type parameters Φ are fully symmetrical to parameters P.

```
template <class T> T *before() {
  return (T *)malloc(sizeof(T));
}

template <class T> T *after() {
  return new T;
}
```

Listing 4: An example of a type-parametrized rule

Listing 4 demonstrates a rule parametrized with type.

5 Results

In this section, we describe how we test `nobrainer`, provide some transformation rule examples and present the performance results.

5.1 Testing

Our tests can be divided into two main groups. First, we have a group of unit- and integration-tests for each phase described in Section 3. These are mainly used to check the correctness of AST matcher generation (Section 4.4) and format string generation (Section 4.4) for various AST nodes.

Second, we have a group of regression tests consisting of several open source projects.

For each project, we have created files with predefined `nobrainer` templates. Our testing framework runs the tool, measures the execution time, checks that all the predefined transformations have been performed as expected, and verifies that the project can be compiled afterwards.

5.2 Examples

In this section, we present three notable transformation rules that are supported by `nobrainer`.

The first example (Listing 5) shows the transformation rule that changes the order of arguments inside of the `compose` method call. Specifically, `nobrainer` will replace each call of the `compose` method of the `Agent` class `a.compose(x, y)` with the call `a.compose(y, x)`.

Thus, we demonstrate how to perform an argument swap automatically when method's signature changes.

```
int NOB_BEFORE_EXPR(ChangeOrder)(Agent a, char *x, char *y) {
  return a.compose(x, y);
}

int NOB_AFTER_EXPR(ChangeOrder)(Agent a, char *x, char *y) {
  return a.compose(y, x);
}
```

Listing 5: An example template for the argument swap

The second example (Listing 6) shows that `nobrainer` can be used to perform simplifying code transformations.

```
class EmptyCheckRefactoring : public nobrainer::ExprTemplate {
public:
  bool beforeSize(const std::string x) {
    return x.size() == 0;
  }

  bool beforeLength(const std::string x) {
    return x.length() == 0;
  }

  bool after(const std::string x) {
    return x.empty();
  }
};
```

Listing 6: An example template for a string emptiness check

Recall that each rule can have an arbitrary number of *before* templates, but only one *after* template. Writing several *before* expressions helps to group common transformations.

The third example contains type and expression parameters. Listing 7 shows the corresponding rule.

```
class ConstCastRefactoring : public nobrainer::ExprTemplate {
public:
  template <class T>
  T *before(const T *x) { return (T *)x; }

  template <class T>
  T *after(const T *x) { return const_cast<T *>(x); }
};
```

Listing 7: An example template for const casts

It detects the C-style cast that "drops" the `const` qualifier from the pointed type and replaces it with an equivalent C++-style cast. Parameter `x` should be of any pointer-to-const type and should be cast to exactly this type, but without a `const` qualifier. `Nobrainer` captures all of these connections and processes them as expected.

5.3 Performance

To measure the performance we run our regression tests five times on a machine with Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz CPU, and 64GB of RAM. The machine runs on Ubuntu 16.04 LTS. We perform the execution in eight threads.

Table 1 contains the results. For each project, we list its size in *lines of code*, the number of replacements `nobrainer` applies during the test, and our time measurements. We distinguish two stages of `nobrainer`'s workflow and measure them separately. The first stage incorporates the project's source code parsing. The second stage contains all the remaining computations up to replacement generation. We divide the whole process this way because the parsing process is performed by the *Clang* framework. For this reason, we can only minimize the time `nobrainer` spends in the second stage.

Project	KLOC	Replacements	Parsing time (s)	Remaining operation time (s)
CMake	493	24	31.36	7.13
curl	129	7	3.17	2.01
json	70	7	13.99	1.34
mysql	1170	10	9.54	3.12
protobuf	264	8	16.62	2.97
v8	3055	6	281.57	28.52
xgboost	43	14	6.75	1.18

Table 1. Performance results

It should be noted that the execution time does not directly correlate with the project's size. Other factors, such as translation unit sizes may also influence the overall performance.

As can be seen in Table 1, the elapsed time varies significantly between projects. In particular, this behavior applies both to the parsing time and to remaining processing time. Therefore, comparing the elapsed time of different projects offers few insights. Thus, in our evaluations, we consider the percentage of time that the file parsing takes from the whole process. Then, we compare this proportion among different projects. Fig. 5 demonstrates the corresponding rates for the regression projects. Our results show that parsing takes up more than 81% of the whole execution time on average. For a global refactoring, all files must be parsed. The fact that the remaining procedure takes less than 20% of the execution time means that `nobrainer` has reached near-optimal performance.

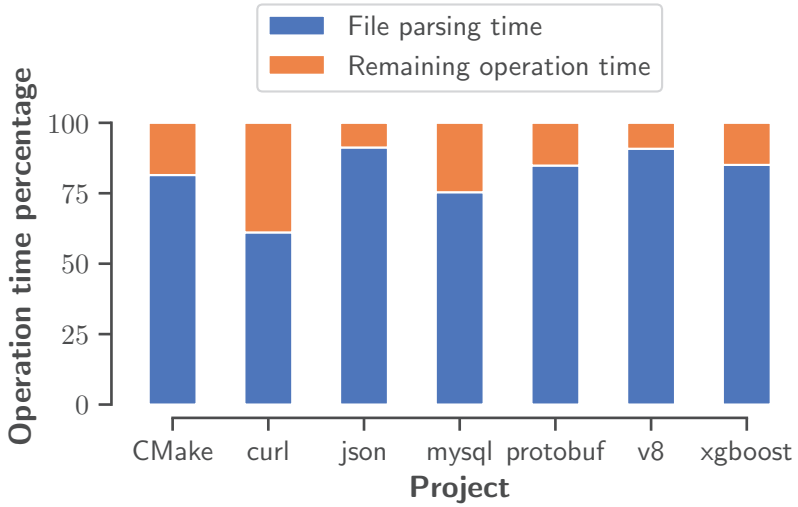


Fig. 5. File parsing percentage in `nobrainer` operation

Nevertheless, in certain cases, it is possible to avoid parsing files when it is sure that the file contains nothing to transform. The next section explains this and other directions in our future work.

6 Limitations and future work

Currently `nobrainer` supports expression templates and type parameterization. It can be used to perform transformations in continuous integration environments (CI). However, the execution time is still unsuitable for running it on

large projects as a background task in IDE. There is still room for improvement. Thus, we consider three main directions for future work:

1. Full statement support
2. Performance improvements
3. Usability improvements

At the moment, we have already designed infrastructure for statement support. This includes API, validation and stubs for processing `Before` and `After` templates. We have also added support for `if` statements, `compound` statements, and variable declaration nodes. Our next task is to implement processing for each remaining statement node.

Regarding performance, we plan to research methods for reducing the parsing time. We are considering two directions. Firstly, we would like to improve the matching phase by skipping files that do not contain symbols used in `Before` templates. Secondly, we will explore automatic precompiled header (PCH) creation, which is expected to speed up the process of parsing the project's header files.

Further, the usability of our tool can be improved in two ways. Currently `nobrainer` performs found transformations only for the whole code base. We would like to add support for executing on user-defined parts of the project. We are also considering integrating with other developer tools. For example, `nobrainer` can be used as an IDE plugin to enhance user experience and the convenience of usage. Another possible scenario is to use `nobrainer` to assist static analyzers for fixing errors or defects.

7 Conclusion

In this paper, we presented `nobrainer` — a transformation and refactoring framework for C and C++ languages based on the Clang infrastructure. Its design is built on two main principles: ease-of-use and the extensive validation of transformation rules. A substantial part of this article includes describing its design and implementation, accompanied with examples and results.

Our results showed that `nobrainer` already supports real-world transformation examples and can be successfully applied to large C/C++ projects in continuous integration environments. We also highlighted the current limitations of the tool and some directions for later improvements. In the future, we plan to enhance the usability of `nobrainer` and integrate with other developer tools.

References

1. Clang documentation: Matching the clang ast. `https://clang.llvm.org/docs/LibASTMatchers.html`

2. Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I., Sangwan, R., Seaman, C., Sullivan, K., Zazworka, N.: Managing technical debt in software-reliant systems. In: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research. pp. 47–52. FoSER '10, ACM, New York, NY, USA (2010). https://doi.org/10.1145/1882362.1882373, http://doi.acm.org/10.1145/1882362.1882373
3. Cunningham, W.: The wycash portfolio management system. SIGPLAN OOPS Mess. **4**(2), 29–30 (Dec 1992). https://doi.org/10.1145/157710.157715, http://doi.acm.org/10.1145/157710.157715
4. Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional (June 1999)
5. G. Waddington, D., Yao, B.: High-fidelity c/c++ code transformation. Electronic Notes in Theoretical Computer Science **141**, 35–56 (01 2007). https://doi.org/10.1016/j.entcs.2005.04.037
6. Graf, E., Zgraggen, G., Sommerlad, P.: Refactoring support for the c++ development tooling. In: OOPSLA Companion (2007)
7. Lahoda, J., Bečička, J., Ruijs, R.B.: Custom declarative refactoring in netbeans: Tool demonstration. In: Proceedings of the Fifth Workshop on Refactoring Tools. pp. 63–64. WRT '12, ACM, New York, NY, USA (2012). https://doi.org/10.1145/2328876.2328886, http://doi.acm.org/10.1145/2328876.2328886
8. Meyerovich, L.A., Rabkin, A.S.: Empirical analysis of programming language adoption. SIGPLAN Not. **48**(10), 1–18 (Oct 2013). https://doi.org/10.1145/2544173.2509515, http://doi.acm.org/10.1145/2544173.2509515
9. Murphy-Hill, E.R., Parnin, C., Black, A.P.: How we refactor, and how we know it. In: ICSE. pp. 287–297. IEEE (2009), http://dblp.uni-trier.de/db/conf/icse/icse2009.html#Murphy-HillPB09
10. Pinto, G.H., Kamei, F.: What programmers say about refactoring tools?: An empirical investigation of stack overflow. In: Proceedings of the 2013 ACM Workshop on Workshop on Refactoring Tools. pp. 33–36. WRT '13, ACM, New York, NY, USA (2013). https://doi.org/10.1145/2541348.2541357, http://doi.acm.org/10.1145/2541348.2541357
11. Ray, B., Posnett, D., Devanbu, P., Filkov, V.: A large-scale study of programming languages and code quality in github. Commun. ACM **60**(10), 91–100 (Sep 2017). https://doi.org/10.1145/3126905, http://doi.acm.org/10.1145/3126905
12. Tracz, W.: Refactoring for software design smells: Managing technical debt by girish suryanarayana, ganesh samarthyam, and tushar sharma. ACM SIGSOFT Software Engineering Notes **40**(6), 36 (2015), http://dblp.uni-trier.de/db/journals/sigsoft/sigsoft40.html#Tracz15a
13. Wasserman, L.: Scalable, example-based refactorings with refaster. In: Proceedings of the 2013 ACM workshop on Workshop on refactoring tools. pp. 25–28. ACM (2013)
14. Wright, H., Jasper, D., Klimek, M., Carruth, C., Wan, Z.: Large-scale automated refactoring using clangmr. In: Proceedings of the 29th International Conference on Software Maintenance (2013)

Deductive proof for industrial applications

Vassil Todorov[1], Safouan Taha[2], Frédéric Boulanger[2] and Armando Hernandez[1]

[1] Groupe PSA, France
[2] LRI, CentraleSupélec, Université Paris-Saclay, France

Abstract. Automotive embedded systems are increasingly more complex and heterogeneous but they are required to be safe, reliable and autonomous. In the near future, self-driving cars are expected to be produced and the authorities would probably require their certification in order to achieve a higher confidence level. Theoretical research and experience show that when using conventional design approaches it is impossible to guarantee high confidence to those systems. The way taken by some industries (e.g. aerospace, railway, nuclear) in order to achieve this higher level of confidence was by using formal verification techniques. One of those techniques is called Deductive proof. It can give a higher level of confidence in proving critical properties that cannot be proved by model checking or for library functions shared by multiple projects.
In this paper, we share our experience about the application of Deductive proof for industrial applications. We show the limitations of the current approaches and discuss some possible solutions. Our discussion is backed up by an example for calculating a square root using linear interpolation.

Keywords: Software verification · Formal methods · Deductive proof · SMT solving

1 Introduction and Motivation

The automotive industry relies mostly on a model-based approach for developing embedded software. It consists in connecting common library blocks (operators) to design and simulate a model of the behavior to be produced. It uses a higher level of abstraction than the code. Code with the behavior of the model is then produced automatically. The most common tools used today are Simulink, from the MathWorks, and Scade, from ANSYS.

Sometimes, the provided library blocks are not sufficient and the designer can import external operators. For our use case, the external operator is a function written in the C programming language calculating a square root using linear interpolation. Lookup tables using linear interpolation are frequently used in automotive embedded software, as they need less processing power. Using deductive proof, our goal is to prove this function correct and thus provide it as a library to the designers. We expect that in the near future, authorities would require that highly critical software for self-driving cars would be certified. This means that we should be able to provide proofs of correction for produced code.

In a previous paper [12], we summarized our experience about applying tools that use formal methods on industrial software. In this paper, we give details about our Deductive method application experience, the problems we encountered and how these problems were solved. Our function was implemented in C and SPARK (based on Ada), and we used Frama-C WP [9] and GNATprove [3] to prove its correctness.

2 Tools for deductive reasoning

As we are interested in tools used by the industry, we present here only those that are mostly used today: Atelier B[3], Caveat [11], Frama-C WP and GNATprove.

Atelier B. Atelier B is a tool supporting the B method, which is a formal methodology to specify, build and implement software systems. The B method was originally developed in the 1980s by Jean-Raymond Abrial [1] and is based on first-order logic, set theory, abstract machine theory and refinement theory.

Caveat and Frama-C WP. Caveat and Frama-C WP are tools for deductive reasoning over C programs. Caveat was introduced at Airbus in 2002 to replace unit tests by unit proof and thus obtain a cost reduction and quality improvement over this part of the development process. The tool with its back-end Alt-Ergo were certified and recognized by the aviation certification authorities. Caveat analyzed C programs (with some restrictions in terms of language constructs) and had its own specification language based on a first order logic. Frama-C is the academic open source tool developed by the same team as Caveat. Its WP module verifies properties written in the ACSL[4] language in a deductive manner. It implements the Weakest Precondition calculus and targets multiple automatic solvers via the Why3 platform[5].

GNATprove. GNATprove is a tool for deductive reasoning over SPARK (based on Ada) programs. It implements the Weakest Precondition calculus and uses the Why3 platform to target multiple automatic provers. If a property cannot be proved, it brings a useful counterexample.

3 Experiment

We take a part of existing production C code and would like to investigate the use of deductive proof in our development process. This code calculates the square

[3] Atelier B: `https://www.atelierb.eu`
[4] ACSL specification language: `https://frama-c.com/acsl.html`
[5] Why3: `http://why3.lri.fr/`

root Y of X from 0.00 to 100.00 by linear integer interpolation between two known points (X_a, Y_a) and (X_b, Y_b) using the following formula:

$$Y = Y_a + (X - X_a)\frac{(Y_b - Y_a)}{(X_b - X_a)}$$

This function is shared in a library, so we want to prove that the calculus is correct for a given precision (between -0.3 and 0.1) before distributing it to the software designers. For our experiment, we used Frama-C with its WP module on our original code, and rewrote the function in SPARK (special thanks to Yannick Moy from AdaCore) to compare different approaches in using weakest precondition for proving properties. We did not use the B method because it is more suitable for the development of control-based modules, not calculus ones.

We proceeded in two steps. Firstly, we proved correct a simplified function containing only eight values in the interpolation table. After proving it correct, we extended our table to 41 values, which is representative of the production code. We present on the next figures the complete code we had to prove. All the values of the interpolation table are not listed for compactness. Fig. 1 shows the interpolation function with its annotated code using ACSL for Frama-C. We calculate the square root of a number between 0.00 and 100.00 using an integer representation because of a hardware limitation. We consider it as a fix-point number multiplied by 100, thus the input range is between 0 and 10000 and the returned result is a linearly interpolated value between 0 and 1000 (to be interpreted as a number between 0.00 and 10.00).

Clauses labeled with `requires` represent the preconditions and those labeled with `ensures` are the postconditions of the proof. The specification or the contract to be proved is written at a function level in special commented lines, which

```
typedef unsigned short uint16;
typedef unsigned char uint8;
/*@ requires 0 <= Xa <= 10000 && 0 <= Xb <= 10000;
 requires 0 <= Ya <= 1000 && 0 <= Yb <= 1000;
 requires Yb > Ya && Xb >= Xa;
 requires Xa <= X <= Xb;
 ensures Xa != Xb ==> \result == (Ya + (X - Xa) * (Yb - Ya) / (Xb - Xa));
 ensures Xa == Xb ==> \result == Ya;
 assigns \nothing;
*/
uint16 LinearInterpolation(uint16 Xa, uint16 Ya, uint16 Xb, uint16 Yb,
    uint16 X) {
   if (Xa != Xb) {
      return(Ya + (X - Xa) * (Yb - Ya) / (Xb - Xa));
   } else {
      return(Ya);
   }
}
```

Fig. 1. Annotated interpolation function for Frama-C WP automatic proof

are ignored by the compiler but analyzed by Frama-C. Preconditions are defined from the user specification (it shall calculate an interpolation for numbers between 0 and 10000, yielding results between 0 and 1000, or return an exact value if it is known). For this function, the postconditions look simple but the proof is not easy. Actually, computer programs use modular arithmetic and the difficulty is to prove that the mathematical formula in the postcondition has the same behavior as the one in the code. For example, we need to prove that the formula in the code does not produce an overflow and that code behaves as math with the given preconditions. The interpolation function is called by the *IntSqrt* function that we present on Fig. 2. This function has two behaviors specified in ACSL: one for numbers between 0 and 10000 and one for numbers greater than 10000. For the first behavior, the square root is calculated by interpolation between the bounds of the interval that contains the provided *number*. The second behavior is for numbers greater than the maximum value of the interpolation table. We consider returning the square root for the maximum value we know i.e. 1000 in our case.

We prove $number - 30 \leq \frac{result^2}{100} \leq number + 10$ to take into account the precision specified. As there is a loop in this function, we need to provide the

```
/*@ assigns \nothing;
 behavior in_range:
 assumes number <= 10000;
 ensures number-30 <= (\result)*(\result)/100 <= number+10;
 behavior out_of_range:
 assumes number > 10000;
 ensures \result == 1000;
 complete behaviors in_range, out_of_range;
 disjoint behaviors in_range, out_of_range;
*/
uint16 IntSqrt(uint16 number) {
   uint8 i = 0;
   uint16 TabX[41] = {0,5,10,25,40,...,7500,8000,8600,9200,10000};
   uint16 TabY[41] = {0,22,32,50,63,...,866,894,927,959,1000};
   /*@ loop invariant 0 <= i <= 40 && number >= TabX[i];
    loop assigns i;
    loop variant 40-i; */
   for (i = 0 ; i < 40 ; i++) {
      if ((number >= TabX[i]) && (number <= TabX[i+1])) {
         return(LinearInterpolation(TabX[i], TabY[i], TabX[i+1],
              TabY[i+1], number));
      }
   }
   return TabY[40];
}
```

Fig. 2. Annotated square root function for Frama-C WP automatic proof

prover an `invariant` for it. This invariant is a clause that is true before, during and after the loop. We also need to indicate a `variant` i.e. a decreasing positive function for this loop to prove its termination.

The equivalent SPARK code is presented on Fig. 3. The main difference with the C programming language is that in SPARK we can specify a bit-vector data type. For our use case, it is more convenient to use bit-vectors for reasoning over modular arithmetic because most SMT solvers used as back-end via Why3 have a theory of bit-vectors. GNATprove has also a feature to return a counterexample when a contract fails, what can be useful for debugging. Basically, it is based on the model that can be obtained by the SMT solver. Actually, the SMT solver is called on the negation of the formula to be proved. If the answer is *"unsat"* the formula is valid. If the answer is *"sat"* then the formula is not valid and getting its model gives a counterexample.

```
type Unsigned is mod 2**32;
subtype uint16 is Unsigned range 0 .. 65535;
type UINT16_ARR is array (Positive range <>) of uint16;
Max : constant := 10_000;
function LinearInterpolation(Xa, Ya, Xb, Yb, X : uint16) return uint16 is
   Result : uint16;
begin
   if Xa /= Xb then
      Result := Ya + (X - Xa) * (Yb - Ya) / (Xb - Xa);
   else
      Result := Ya;
   end if;
   return Result;
end LinearInterpolation;
function IntSqrt(number : uint16) return uint16
   with Global => null, Contract_Cases =>
   (number <= Max => IntSqrt'Result * IntSqrt'Result / 100 + 30 >=
       number and number+10 >= IntSqrt'Result * IntSqrt'Result / 100,
       number > Max => IntSqrt'Result = 1000) is
   TabX : UINT16_ARR(1 .. 41) := (0,5,10,25,40,...,8000,8600,9200,10000);
   TabY : UINT16_ARR(1 .. 41) := (0,22,32,50,63,...,894,927,959,1000);
begin
   for I in 1 .. 40 loop
      pragma Loop_Invariant (for all J in 1 .. I => number >= TabX(J));
      if number in TabX(I) .. TabX(I+1) then
         return LinearInterpolation (TabX(I), TabY(I), TabX(I+1),
             TabY(I+1), number);
      end if;
   end loop;
   return TabY(41);
end IntSqrt;
```

Fig. 3. SPARK code for automatic proof with GNATprove

Another feature of GNATprove is to do a dynamic contract checking. Function *IntSqrt* is called with all possible values of its parameter *number* and the tool checks the validity for each value of the already present function contracts for deductive proof. The equivalent feature for Frama-C is the E-ACSL plugin[6].

4 Results

Scenarios. Our different scenarios are presented on Fig. 4. The SPARK code using bit-vectors is transformed by GNATprove into Verification Conditions (VCs) that are independent goals to be proved. It generates a WhyML file for Why3. The role of Why3 is to manage the proof using appropriate languages and theories for each prover. Frama-C WP transforms annotated C code into VCs and generates one WhyML file per function. When using Why3 quantifiers are introduced but unfortunately, all SMT solvers do not support them. There was also

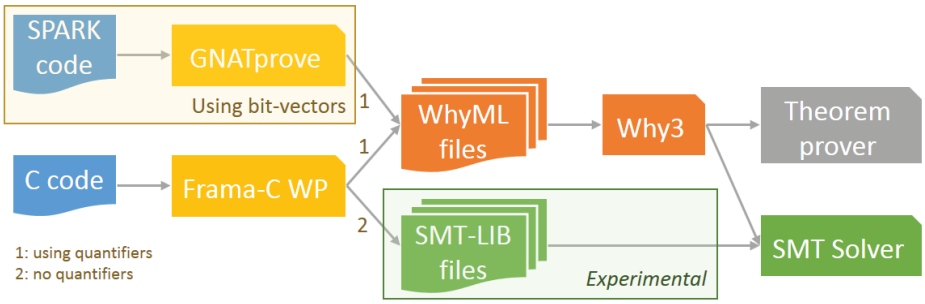


Fig. 4. Using deductive proof on C and SPARK code

an experimental feature in Frama-C providing a direct SMT-LIB output without quantifiers. We used the SMT-LIB output of Frama-C and Why3 to understand the difficulties the prover had when it did not succeed.

Simplified toy example. We began with a simple example using only eight values *(0, 5, 10, 25, 40, 55, 75 and 100)* in the interpolation table. Frama-C and GNATprove proved it correct in a reasonable time (less than 10 seconds).

Real production example. Then we extended the interpolation table to 41 values (ranging from 1 to 10000 instead of 1 to 100). We expected that it would be easy to prove. In reality, the response time increased exponentially. GNATprove proved our code in almost 100 seconds using CVC4 and Z3 thanks to the use of bit-vectors. The Frama-C official version was unable to prove 2 of the 51 goals.

[6] Frama-C E-ACSL: `https://frama-c.com/eacsl.html`

We wanted to understand why scaling from 8 to 41 values in a table generated so much difficulties to the solvers we used. We used different ways to request solvers to prove our most difficult goals – the postconditions of each function. The main difficulty was to prove that the nonlinear interpolation function, returning a 16-bit unsigned integer, does not overflow and has the same behavior as its mathematical equivalent. It is not a trivial problem and few SMT solvers were able to prove it. To do so, they had to use some sort of advanced interval analysis and propagation techniques. We also noticed that it was easier to prove these goals using the quantifier-free formulae approach. For example, CVC4 proved a goal using the SMT-LIB output of Frama-C without quantifiers while it was unable to prove the same goal using the Why3 SMT output with quantifiers. The main difference between the quantifier-free SMT output of Frama-C and the Why3 quantified SMT output is that Why3 redefines operators such as division using uninterpreted functions. To simplify the problem, we removed these specific functions and used the standard SMT `div` operator. Thus, the proof succeeded using a nonlinear logic containing bit-vectors. Disabling bit-vectors from that logic resulted in an impossibility to prove the result. On the other hand, the quantifier-free SMT output did not need bit-vector logic to be proved.

As solvers are complex programs and may contain bugs, we want to validate our goals by at least 2 solvers. In Table 1, we summarize the results obtained for our most difficult goals. The proof approach represents the way to use the provers. We can either use Frama-C and Why3 to access a large number of solvers or directly send specific SMT-LIB requests to the solvers. The first approach introduces quantifiers and uninterpreted functions. The second excludes quantifiers and uses SMT-LIB functions. For our use case, Colibri and CVC4 were the most relevant solvers.

Table 1. Results in seconds for the most difficult goals using Frama-C

Proof approach		IntSqrt_in_range_post	LinearInterpolation_post
Why3	Alt-Ergo	Timeout	26662
	Colibri	Unsupported	Unsupported
	CVC4	unknown	unknown
	Yices2	Unsupported	Unsupported
	Z3	unknown	1278
SMT-LIB	Colibri	2.9	0.3
	CVC4	3165	3.3
	Yices2	Timeout	0.1
	Z3	unknown	unknown

5 Conclusions and Future work

In this paper, we have illustrated an industrial use case about proving the correctness of a square root function using linear interpolation. We used Frama-C WP and GNATprove, which are two industrial tools for deductive formal proof. Even if our function was relatively simple, we found that nonlinear and modular arithmetic are still a challenge for most of the provers. Two non-standard approaches worked well for us: the use of bit-vectors in SPARK and the direct SMT-LIB quantifier-free output of Frama-C. Bit-vectors are well supported in most of the modern SMT solvers and are well suited for problems that involve modular arithmetic but they do not scale up. For our use case, SMT requests without quantifiers performed better.

Using deductive methods is very promising in industrial context for safety-critical applications. It can replace unit tests as shown in [10] and thus decrease cost while increasing quality. It is also an intellectual activity that brings more satisfaction for engineers compared to test activity. We expect that in the future, the competition between solvers will bring a higher level of automation and help with useful counterexamples. We plan to investigate how proof artifacts obtained by deductive tools could be combined with artifacts obtained by model checking and abstract interpretation tools.

References

1. Abrial, J.R.: The B-book : assigning programs to meanings (1996)
2. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanovi'c, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) (CAV'11). LNCS, vol. 6806, pp. 171–177. Springer (Jul 2011)
3. Chapman, R.: Industrial Experience with SPARK. Ada Letters **XX**(4) (2000)
4. Conchon, S., Coquereau, A., Iguernlala, M., Mebsout, A.: Alt-Ergo 2.2. In: SMT Workshop: International Workshop on SMT. Oxford, United Kingdom (Jul 2018)
5. De Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. TACAS'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg (2008)
6. Dijkstra, E.W.: Guarded Commands, Nondeterminacy and Formal Derivation of Programs. Commun. ACM **18**(8), 453–457 (Aug 1975)
7. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification. pp. 737–744. Springer International Publishing, Cham (2014)
8. Hoare, C.A.R.: An Axiomatic Basis for Computer Programming (Oct 1969)
9. Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-C: A software analysis perspective. Formal Aspects of Computing **27**(3) (2015)
10. Moy, Y., Ledinot, E., Delseny, H., Wiels, V., Monate, B.: Testing or Formal Verification: DO-178c Alternatives and Industrial Experience. IEEE Soft. **30**(3) (2013)
11. Randimbivololona, F., Souyris, J., Baudin, P., Pacalet, A., Raguideau, J., Schoen, D.: Applying Formal Proof Techniques to Avionics Software: A Pragmatic Approach. In: Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems. Springer-Verlag, London (1999)
12. Todorov, V., Boulanger, F., Taha, S.: Formal Verification of Automotive Embedded Software. In: Proceedings of the 6th Conference on Formal Methods in Software Engineering. pp. 84–87. FormaliSE '18, ACM, New York, NY, USA (2018)

Providing the sharing of heterogeneous ontology design patterns in the development of the ontologies of scientific subject domains*

Yury Zagorulko and Olesya Borovikova

A.P. Ershov Institute of Informatics Systems, Lavrent'ev av., 6, Novosibirsk 630090, Russia {zagor, olesya}@iis.nsk.su

Abstract. The paper describes an approach to solving the problems of using ontology design patterns (ODPs) for the development of the ontologies of scientific subject domains (SSDs). This approach offers a system of the heterogeneous ODPs, including both universal patterns and patterns oriented to the presentation of scientific knowledge, as well as methods of their joint use for building ontologies of SSDs. The use of this approach allows us to save resources spent on the development of ontologies, avoid errors common in ontological modeling, as well as ensure a consistent presentation of all the entities of the ontologies of scientific subject domains.

Keywords: Ontology · Scientific subject domain · Ontology design pattern · Structural pattern · Content pattern.

1 Introduction

Currently, ontologies are the main means of formalization and systematization of knowledge in various subject areas including scientific subject domains (SSDs). (Note that by "scientific subject domain" we mean a subject area that encompasses a branch of science or field of scientific knowledge in all its aspects.)

The development of ontology is a very complicated and time-consuming process. To simplify and facilitate it, various methods of and approaches to ontology development [1–4] have been proposed. Recently, an approach based on ontology design patterns (ODPs) [5–7] has gained popularity. According to this approach, ODPs are documented descriptions of proven solutions to typical problems of ontological modeling. Despite the fact that the use of ODPs allows us to greatly simplify the process of building ontologies and improves their quality, ontology design patterns have not yet found wide practical application due to a number of problems arising from their use.

One of the widespread problems of pattern reuse is their complexity: it is often difficult for the developer of a new ontology to understand the semantics the authors have laid down in the pattern. Another common problem is that the

* The work was financially supported by the Russian Foundation for Basic Research (Grant No. 19-07-00762).

patterns are described and used separately and do not constitute a single system. In the development of ontologies of SSDs, there is yet another important problem, which is the absence of patterns designed to present scientific knowledge.

The paper presents the approach to the construction of ontologies of scientific subject domains based on the ODPs. The approach complements and develops the ontology development methodology proposed by the authors and used in development of intelligent scientific internet resources [8]. The ODPs used in this approach emerged as a result of solving the problems of ontological modeling, which the authors of the paper encountered in the process of developing ontologies for various scientific subject domains [9, 10].

This paper is organized as follows. The second section contains a short review of the ontology design patterns; the third section analyzes the problems of their use. The proposed approach to the development of ontologies of scientific subject domains is described in detail in the fourth section. The main advantages and practical benefits of this approach, as well as plans for the near future, are discussed in the Conclusion.

2 A Short Review of Ontology Design Patterns

The progenitors of ontology design patterns are design patterns, widely used in software development [11]. Similar to this design patterns, ODPs are employed to describe solutions of typical problems arising in the development of ontologies [7].

Depending on the problems for solution of which the ODPs are created, we distinguish between structural patterns, correspondence patterns, content patterns, reasoning patterns, presentation patterns and lexico-syntactic patterns. (Note that this typology of patterns was proposed in the framework of the NeOn project [12].)

From all types of patterns listed above only structural patterns, patterns of content and presentation are used in the development of ontologies.

The structural patterns either fix the ways to solve problems caused by the limitations of the expressive capabilities of ontology description languages or specify the general (modular) structure of an ontology. Patterns of the first type are called logical patterns, and patterns of the second type are called architectural patterns.

The content patterns define the ways of representing typical ontology fragments, on the basis of which ontologies of a whole class of subject domains can be built.

The presentation patterns actually represent the rules (recommendations) for naming and annotating elements of ontology. The application of these rules should increase the readability of the ontology, as well as the convenience and ease of its use.

Currently, several catalogs of ODPs have been created and are developing [13–15]. The most complete of them is posted on the ODPA (Association for

Ontology Design & Patterns) portal [13], created as part of the NeOn project [12].

ODPs are most often described in the format proposed on the ODPA association portal [13]. According to it, the description of the pattern includes information about its author and scope, its graphical representation, text description, a set of scenarios and examples of usages, and links to other patterns. Content patterns can also be supplied with a set of competency questions [6, 7], which can be used both in the development of patterns and in the search for the desired patterns in the development of a specific ontology.

3 Problems of Using Ontology Design Patterns

The first problem of pattern reuse is due to their complexity: it is often difficult for the developer of a new ontology to understand the semantics that the authors have laid down in the pattern. Recently there has been a tendency to simplify patterns [16]. Even so-called meta-patterns, describing very simple entities, were suggested [17]. However, such simple patterns cannot significantly facilitate the construction of SSD ontologies.

Another problem is caused by the lack of convenient ontology development tools supporting the use of ODPs. Here we can note the plugins for the ontology development tool of the project NeOn [12] and the ontology editor WebProtégé [18]. However, the first plugin is available only to the participants of the NeOn project, and the second can be used only in the WebProtégé editor, which is not yet very popular among ontology developers due to its limited functionality (in comparison with the desktop version).

The third problem is that the patterns are described and applied separately and do not constitute a single system. One more problem associated with this problem is the lack of systematic sets of patterns targeted at subject matter experts. Existing catalogs of ontology design patterns do not meet this requirement.

In our opinion, the OTTR library (Reasonable Ontology Templates) [19] is the closest to solving the latter problem. This library provides a language for the representation of ontology design patterns and software supporting it. The OTTR library supplies ontology developers with patterns in the form of high-level OWL macros, which makes possible their use by subject matter experts.

As for the availability of patterns that can be used in the development of SSD ontologies, the catalogs mentioned above do not even partially cover the needs of building ontologies of scientific fields.

4 Approach to the Development of Ontologies of Scientific Subject Domains

This section describes an approach to solving the problem of reusing ODPs in the development of ontologies of scientific subject domains. This approach offers

a system of heterogeneous ODPs and methods for their sharing (joint use) for building SSD ontology. At the moment, there are three types of patterns in the system: structural logical patterns, content patterns and presentation patterns. One part of these patterns is universal, and the other part is focused on the presentation of scientific knowledge.

An important feature of this approach is the use of base (core) ontologies, which include only the most general entities that are not dependent on a particular SSD. These ontologies were previously developed for the technology for building subject-based intelligent scientific internet resources [8] and are now represented by content patterns which were developed for all main entities of base ontologies. In this regard, the construction of SSD ontology using the base ontologies is reduced to their specialization and expansion. In particular, the content patterns presented in the base ontologies are tuned (specialized) to a specific SSD. As for the population of SSD ontology with actual data, it is performed by instantiation of content patterns. This process is supported by a data editor developed in the frameworks of this approach.

4.1 An SSD Ontology and Base Ontologies

Usually the ontology of any SSD contains not only descriptions of its inherent system of concepts and methods for processing and analyzing information, but also descriptions of relevant information resources. In this regard, an SSD ontology can be represented as a system of interrelated ontologies responsible for representing the above three components of knowledge, namely, the ontology of the knowledge domain, the ontology of tasks and methods, and the ontology of scientific Internet resources.

The ontology of the knowledge domain defines the system of concepts and relations intended for a detailed description of a modeled SSD and its scientific and research activities. The ontology of tasks and methods describes the tasks solved in a given SSD and the methods for their solution. The ontology of scientific Internet resources is used to describe the information resources available on the Internet relevant to this SSD.

Since the development of an ontology of an SSD from scratch is not an easy task, we have proposed a method for its construction based on a small but representative set of base ontologies that include only the most general entities not dependent on a particular SSD. This set includes: (1) the ontology of scientific knowledge, (2) the ontology of scientific activity, (3) the base ontology of tasks and methods, (4) the base ontology of information resources.

As mentioned above, all base ontologies have specifications in the OWL language [21].

The ontology of scientific knowledge contains classes that define structures for describing concepts included in any SSD. Such concepts are *Division of science*, *Object of research*, *Subject of research*, *Method of research*, *Scientific result*, etc.

The ontology of scientific activity includes classes of concepts related to the organization of research activities, such as *Person*, *Organization*, *Event*, *Activity (Scientific activity)*, *Project*, *Publication*, etc.

The base ontology of information resources includes the class *Information resource* as the main class. The set of properties (attributes and relationships) of this class is based on the Dublin core standard [20].

Concepts and relations of base ontology of tasks and methods are used to describe tasks to be solved in a given SSD, methods for their solution and software components and algorithms implementing them.

4.2 A System of Ontology Design Patterns

To support the considered approach, a set of ODPs [21] was developed and implemented in the OWL language [22]. This set includes various types of patterns: structural logical patterns, content patterns and presentation patterns. All these patterns are combined into a single system.

Note that in this approach the presentation patterns define the rules for naming and annotating elements of ontology, which are close to the generally accepted ones [23].

The need to use structural logical patterns was attributed to the absence in OWL of expressive means for representing complex entities and structures required for building SSD ontologies, in particular, the ranges of admissible values, and n-ary and attributed relations (a binary relation with attributes).

The pattern of representation of the range of admissible values is intended to specify such structures that are called domains in the relational data model and are characterized by a name and a set of elementary values. Domains are convenient to use for describing possible values of class properties when the entire set of such values is known in advance. In this pattern, the domain is defined by an enumerated class, which is the successor of the specially introduced service class called the Domain class and consists of a finite set of different individuals (objects) determining the possible values of a certain property (see Fig. 1).

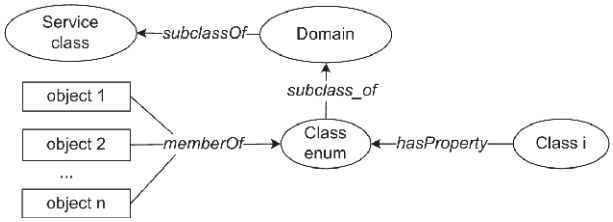


Fig. 1. Structural pattern of representation of the range of admissible values.

Examples of such domains are “Geographic type”, “Position”, “Type of organization”, “Type of publication”, which include, respectively, types of localities, types of positions in organization, types of organizations and publications.

Note that in the figures of the patterns presented in the paper, classes are shown in the form of ellipses, individuals and attributes are in the form of rectangles. An *ObjectProperty* type connection (a relation) is shown by a solid straight line, and a *DataProperty* type connection (an attribute), by a dash line. At the

same time, classes, attributes and individuals, which must necessarily be present in the pattern, are represented by figures surrounded by a thick line.

To represent an attributed relation, a structural pattern is proposed. It is shown in the left side of Fig. 2.

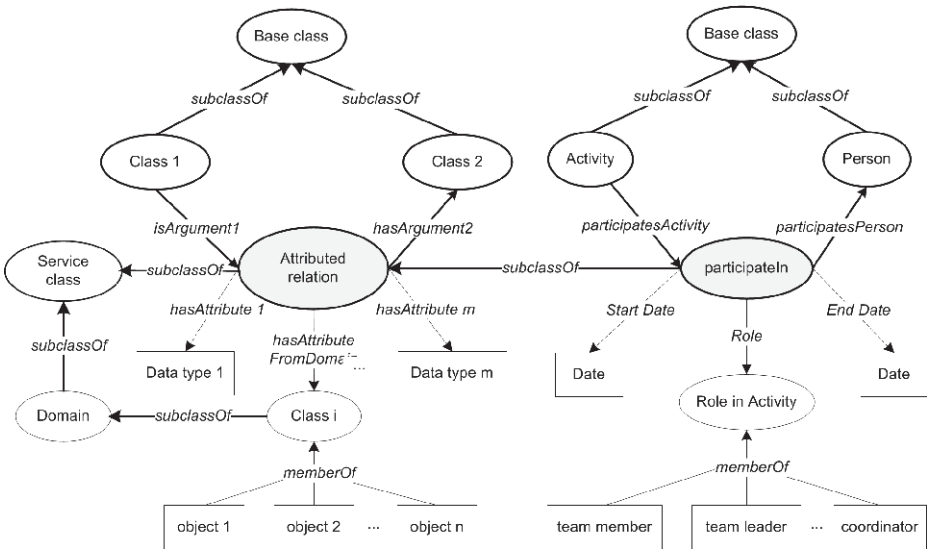


Fig. 2. Structural pattern of the binary attributed relation and an example of its specialization.

The central place in this pattern is occupied by the service class *Attributed relation* with which the base classes of an ontology modeling the arguments of the binary relation are connected by the links *isArgument1* and *hasArgument2*. At the same time, the attributes of a binary relation are modeled by the properties of this class (in OWL notation, either *DataProperty* or *ObjectProperty*) *hasAttribute* and *hasAttributeFromDomain*. For this pattern, it is required to set constraints on the obligatoriness and uniqueness of the arguments of the attributed relation (*Class 1* and *Class 2*).

To represent a specific type of the attributed relation, a new class, which is its successor, can be defined.

The right side of Figure 2 shows an example of a structural pattern for describing a person's participation in scientific activities (the attributed relation *participateIn*). Here, the *Person* class serves as the first argument, the *Activity* class is the second argument. The pattern also allows us to specify the start and end dates of the person's participation in an activity, as well as his/her role in it.

Similarly, we can build a pattern for an n-ary relation. Note that for this pattern we must also specify the order of the arguments.

For a uniform and consistent presentation of the concepts used in SSD and their properties, content patterns were constructed for the main concepts of base ontologies using the structural patterns proposed. Due to this, the development of an ontology of a specific SSD mainly consists in the specialization of content patterns and the construction of fragments of a target ontology based on them.

As an example, we give a pattern intended for the description of applied tasks solved within the framework of a scientific subject domain (see Fig. 3).

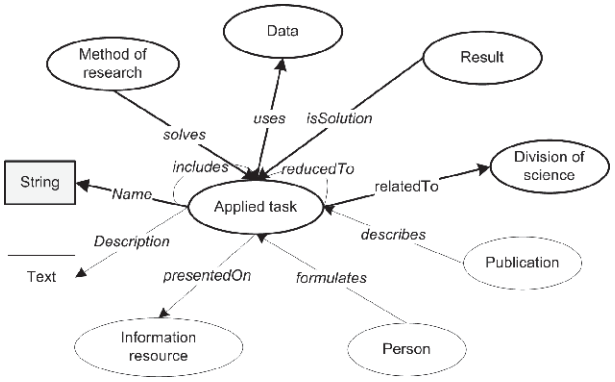


Fig. 3. Pattern for describing the applied task.

The following set of competency questions represents the content of this pattern:

What methods solve the applied task?

What data is used for solving the applied task?

What is the result of solving the applied task?

Who formulates the task?

and etc.

It should be noted that the content patterns included in the proposed set are interrelated through common concepts and relationships and thus form a single network of patterns. For example, presented in Fig. 4 content patterns, describing the concepts of *Activity* and *Person*, are interconnected not only by the attributed relation *participateIn*, but also through the concepts of *Scientific result*, *Method of research*, *Publication*, and *Organization*.

Note that in the Fig. 4 the attributed relations *participateIn* and *workIn* are shown by a dotted line.

4.3 Methods of Building Ontologies of SSDs

Building an SSD ontology involves two main steps:

1. Construction of the components of SSD ontology using the base ontologies through their specialization and expansion.
2. Population of SSD ontology with actual data by instantiation of content patterns presented in base ontologies and specialized at step 1.

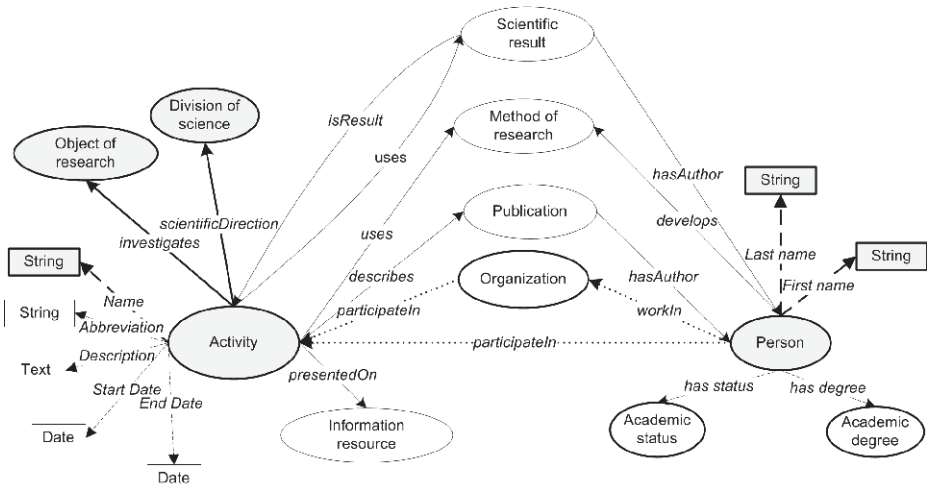


Fig. 4. Fragment of a network of patterns.

Note that in this approach the ontology of the knowledge domain is built on the basis of ontologies of scientific knowledge and scientific activity; ontology of tasks and methods, on the basis of base ontology of tasks and methods; and ontology of scientific Internet resources, on the basis of base ontology of Internet resources.

The use of content patterns is supported by a special editor, which allows specialists in the subject area to populate the ontology with actual data, i.e. objects of classes and their properties. When populating an ontology with the help of the editor, the user selects the required class from the class hierarchy presented to him, and the editor uses the class name to find the corresponding pattern. After that, the editor, using the information from the pattern, builds a form containing the fields for filling in all the properties of the object of this class. At the same time, the editor can interpret the relations with attributes described by the patterns. Thanks to this, the user can work with the properties of the created object that are set by such relations as with "ordinary" object properties. The difference consists only in the need to specify the values of the attributes in a separate window.

5 Conclusion and Future work

The paper discusses the problems of applying ontological design patterns for the development of ontologies of scientific subject domains. An approach to the development of SSD ontologies that solves most of these problems is presented. This approach is supported by a system of heterogeneous ontology design patterns, describing the main structures and entities necessary for describing scientific domains, and the data editor, which makes it possible to populate the ontology

with actual data by instantiation of content patterns. Due to the simplicity and clarity of the pattern system and the data editor, this approach can be used not only by knowledge engineers, but also by specialists in the modeled area of knowledge.

This approach has shown its practical utility in the development of ontologies of various scientific subject domains ("Decision Support" [24], "Active Seismology" [25], etc.).

In the near future, it is planned to expand this approach in such a way that it provides automated population of ontology. For this, the pattern system will be expanded with lexico-syntactic patterns [26], which will be used to facilitate the population (completion) of ontologies based on texts in the natural language. Lexico-syntactic patterns are supposed to be automatically generated based on the existing content and structural patterns using the synonyms dictionary and subject area thesaurus.

References

1. Fernández-López, M., Gómez-Pérez, A., Pazos, A., Pazos, J. Building a Chemical Ontology Using Methontology and the Ontology Design Environment. IEEE Intelligent Systems & their applications 4(1), pp. 37–46 (1999).
2. Sure, Y., Staab, S., Studer, R.: Ontology Engineering Methodology. In: Staab S., Studer R. (eds.) Handbook on Ontologies. pp. 135–152. Springer Verlag, Berlin (2009).
3. Pinto, H., Staab, S., Tempich, C.: DILIGENT: Towards a fine-grained methodology for DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies. In: the proceedings of the 16th European Conference on Artificial Intelligence. Frontiers in Artificial Intelligence and Applications, 110, pp. 393–397. IOS Press, Amsterdam (2004).
4. De Nicola, A., Missikoff, M., Navigli, R.: A Proposal for a Unified Process for Ontology Building: UPON. In: Andersen K.V., Debenham J., Wagner R. (eds) Database and Expert Systems Applications. DEXA 2005. Lecture Notes in Computer Science, vol. 3588, pp. 655–664. Springer, Berlin, Heidelberg (2005).
5. Gangemi, A., Presutti, V.: Ontology Design Patterns. In: Staab, S., Studer, R. (eds.) Hand-book on Ontologies, International Handbooks on Information Systems, pp. 221–243. Springer-Verlag, Berlin, Heidelberg (2009).
6. Blomqvist, E., Hammar, K., Presutti, V.: Engineering Ontologies with Patterns: The eXtreme Design Methodology. In: Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., Presutti, V. (eds.) Ontology Engineering with Ontology Design Patterns, Studies on the Semantic Web, vol.25, pp. 23–50. IOS Press, Amsterdam (2016).
7. Karima, N., Hammar, K., Hitzler, P.: How to Document Ontology Design Patterns. In: Hammar, K., Hitzler, P., Krisnadhi, A., awrynowicz, A., Nuzzolese, A., Solanki, M. (eds.) Advances in Ontology Design and Patterns, Studies on the Semantic Web, vol.32, pp. 15–28. IOS Press, Amsterdam/AKA Verlag, Berlin (2017).
8. Zagorulko, Y., Zagorulko, G.: Ontology-Based Technology for Development of Intelligent Scientific Internet Resources // Intelligent Software Methodologies, Tools and Techniques. Proceedings of 14th International Conference, SoMet 2015. Hamido Fujita, Guido Guizzi (Eds.), Communications in Computer and Information Science, Vol. 532, Springer International Publishing, 2015.pp. 227–241.

9. Zagorulko, Yu., Borovikova, O.: Technology of Ontology Building for Knowledge Portals on Humanities. In: Wolf, K.E. et al.(eds.) Knowledge Processing and Data Analysis. KONT/KPP 2007. Lecture Notes in Artificial Intelligence, vol. 6581, pp. 203–216. Springer-Verlag Berlin, Heidelberg (2011).
10. Borovikova, O., Globa, L., Novogrudska, R., Ternovoy, M., Zagorulko, G., Zagorulko, Yu.: Methodology for knowledge portals development: background, foundations, experience of application, problems and prospects. Joint NCC & IIS Bulletin, Series Computer Science (34), 73–92 (2012).
11. Johnson, R., Vlissides, J., Helm, R.: Design Patterns: Elements of Reusable Object-Oriented Software by Erich Gamma. Addison-Wesley Professional, Boston (1994).
12. NeOn Project, http://www.neon-project.org. Last accessed 2 Feb 2019.
13. Association for Ontology Design & Patterns, http://ontologydesignpatterns.org. Last accessed 2 Feb 2019.
14. Ontology Design Patterns (ODPs) Public Catalog, http://odps.sourceforge.net. Last accessed 2 Feb 2019.
15. Dodds, L., Davis, I.: Linked Data Patterns (2012), http://patterns.dataincubator.org/book. Last accessed 2 Feb 2019.
16. Krisnadhi, A., Hitzler, P.: A Core Pattern for Events. In: Hammar, K., Hitzler, P., Krisnadhi, A. (eds.) Advances in Ontology Design and Patterns, 32. IOS Press, Kobe, Japan, pp: 29–37 (2017).
17. Krisnadhi, A., Hitzler, P.: The Stub Metapattern. A Core Pattern for Events. In: Hammar, K., Hitzler, P., Krisnadhi, A. (eds.) Advances in Ontology Design and Patterns, 32. IOS Press, Kobe, Japan, pp: 29–45 (2017).
18. Hammar, K.: Ontology Design Patterns in WebProtg. In the proceedings of 14th Interna-tional Semantic Web Conference (ISWC-2015). Posters & Demonstrations Track. CEUR Workshop Proceedings, 1486, (2015) http://ceur-ws.org/Vol-1486/paper_50.pdf. Last accessed 2 Feb 2019.
19. Skjveland, M.G., Forssell, H., Klwer, J.W., Lupp, D., Thorstensen, E., Waaler, A.: Pattern-Based Ontology Design and Instantiation with Reasonable Ontology Templates. In: the proceedings of the 8th Workshop on Ontology Design and Patterns (WOP 2017). Vienna, Austria, October 21. CEUR Workshop Proceedings, 2043, (2017) http://ceur-ws.org/Vol-2043/paper-04.pdf. Last accessed 2 Feb 2019.
20. DCMI Metadata Terms, http://dublincore.org/documents/dcmi-terms. Last accessed 2 Feb 2019.
21. Zagorulko, Y., Borovikova, O., Zagorulko, G.: Development of Ontologies of Scientific Subject Domains Using Ontology Design Patterns. In: Kalinichenko, L., Manolopoulos, Y., Skvortsov, N., Sukhomlin, V. (eds.) Selected Papers of the XIX International Conference on Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2017), Communications in Computer and Information Science, 822, pp. 141–156. Springer, Heidelberg (2018).
22. Antoniou, G., Harmelen, F.: Web Ontology Language: OWL. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, pp. 91–110. Springer Verlag, Berlin (2009).
23. Noy, N., McGuinness, D.: Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001. Stanford (2001).
24. Zagorulko, Yu., Zagorulko, G.: Features of Development of Internet Resource for Supporting Developers of Intelligent Decision Support Systems. In the proceedings of Eight International conference “Open Semantic Technologies for Intelligent Systems” pp. 63–66. Belarus, Minsk. (2018).

25. Braginskaya, L., Kovalevsky, V., Grigoryuk, A., Zagorulko, G.: Ontological approach to information support of investigations in active seismology. In: the proceedings of the 2nd Russian-Pacific Conference on Computer Technology and Applications (RPC),Vladivostok, Russky Island, Russia, 25-29 September, pp. 27–29, IEEE Xplore digital library (2017). http://ieeexplore.ieee.org/document/8168060. Last accessed 2 Feb 2019.
26. Maynard, D., Funk, A., Peters, W.: Using lexico-syntactic ontology design patterns for ontology creation and population. In Proceedings of WOP2009 collocated with ISWC2009, vol. 516. pp. 39–52. CEUR-WS.org (2009), http://ceur-ws.org/Vol-516/pap08.pdf. Last accessed 2 Feb 2019.

Effective Scheduling of Strict Periodic Task Sets with Given Permissible Periods in RTOS

S.A. Zelenova[1] and S.V. Zelenov[1,2][0000−0003−0446−0541]

[1] Ivannikov Institute for System Programming of the Russian Academy of Sciences (ISP RAS)
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
{sophia,zelenov}@ispras.ru
http://www.ispras.ru/
[2] National Research University Higher School of Economics (HSE)
20, Myasnitskaya Ulitsa, Moscow, 101000, Russia
szelenov@hse.ru
https://www.hse.ru/

Abstract. In the paper, we suggest new approach to schedulability problem for strict periodic tasks (a periodic task is strict if it must be started in equal intervals of time – task's period). Given permissible tasks' periods, our approach allows to obtain quickly all schedulable sets of tasks with such periods and to build immediately a conflict-free schedule for each obtained set. The approach is based on mathematical methods of graph theory and number theory. We illustrate the approach by a number of examples and present current practical results.

Keywords: scheduling · real-time system · strict periodic task.

1 Introduction

Real-time systems are complex and promissing area of research. The such kind of system requires distributed computing for real relation representation. Thus it becomes necessary to have different kind of scheduling of task processing.

Now we recall some terms.

Suppose processor time is divided into minimal parts (scheduler quantums) that are numerated. We refer to such a part as a *point*.

A point number t is called *starting* for a task if the task processing starts at this point. The processing points different from the starting point are called *additional*. A task duration is the number of all processing points for the task.

A task is called *periodic*, if its processing is repeated at equal time intervals. Length of time of one such interval is called a *period* p of the task. A periodic task is called *strict periodic*, if its adjacent starting points are at a distance exactly equal to the task period. Besides, additional points related to the same processing must be processed during the period following the corresponding starting point.

In this paper, we discuss static[3] scheduling of strictly periodic preemptive[4] tasks.

There are two simple considerations used to find conflict-free schedules.

The first is CPU usage. Let $p_1, ..., p_k$ be periods of periodic tasks executed on a processor. Let l_i be the worst case execution time for one instance of the task with period p_i. Then CPU usage should not be more than 100% [1]:

$$\frac{l_1}{p_1} + ... + \frac{l_k}{p_k} \leq 1. \tag{1}$$

The second is the necessary condition for conflict-free schedules. Let p_1 and p_2 be periods of two tasks. And let t_1 and t_2 be starting points of the tasks. If $t_1 - t_2$ is divisible by the greatest common divisor (GCD) of p_1 and p_2 without remainder, then the schedules conflict [2].

The standard way to find a schedule for a task system consists of two steps. the *first step* is to place the starting points. The *second step* is to place the additional points. These two actions have to be considered separately, because it is not clear how to distribute the starting and additional points at the same time, and whether it is possible in principle.

So far, scheduling algorithms have been based on the search of suitable points [3–5]. But exhaustive search evokes the problem of combinatorial explosion. The available methods of limiting the search are unsatisfactory, therefore, it is not uncommon when a scheduling algorithm fails to find a solution even though it exists.

We propose a different approach to the problem described above. This approach is based on our study of numerical properties of period systems [6].

Our research concerns only the first step of scheduling, namely, the distribution of starting points. Therefore, in the rest of the paper, the term "schedule" means only the location of starting points (thus, we mean all durations $l_i = 1$ for all tasks). In the paper, we propose an effective algorithm that enumerates all schedulable task sets and builds starting points for them in an acceptable time. We describe an outline of this algorithm and present some experimental results.

2 Preliminaries

Let D be a set of GCD of task period pairs. And let H be a closure of D with respect to the operation of taking the greatest common divisor. Denote by G_H a directed graph constructed as follows. Vertices of G_H are elements of H and there is an edge from a vertice d_1 to a vertice d_2 if d_2 is divisible by d_1 and there is no any d_3 such, that d_3 is divisible by d_1 and d_2 is divisible by d_3. The vertice d_1 is called a *parent* and d_2 is called a *child*.

Now, for all d from H, denote by G_d an undirected graph constructed as follows. Vertices of G_d are all task periods and there is an edge between two vertices if the periods have GCD equal to d.

[3] Scheduling is called *static*, if the schedule is built before running the system.

[4] Task is called *preemptive*, if it may be interrupted by another task

In [6] we prove the following criterion for the existence of a conflict-free schedule:

Theorem 1. *The system of tasks with periods p_1, p_2, ..., p_k has a conflict-free schedule if and only if*

1. *For all $d \in G_H$, graph G_d have the proper coloring involving at most of d colors.*
2. *Colors in different G_d graphs are "inherited" with respect to G_H. This means that two vertices having different colors in G_{parent} for a parent node from G_H cannot have the same color in G_{child} for a child node from G_H.*
3. *For any $d \in G_H$ with parents $d_i \in G_H$, any period subset, which is colored by the same color in each G_{d_i}, is colored by at most $m = \frac{d}{LCM(d_i)}$ colors in G_d. The number m is called multiplier for a divider d.*

Figure 1 shows the case when the construction of a conflict-free schedule is impossible due to a violation of the third condition of theorem 1. Indeed, in order to properly color G_2, the set $S = \{6, 12, 18, 30\}$ must be colored by only color. On the other hand, in order to properly color G_6, this set must be colored by different colors. But $m = \frac{6}{2} = 3$ for the parent 2 and the child 6, thus there are only three colors for coloring four elements of S.

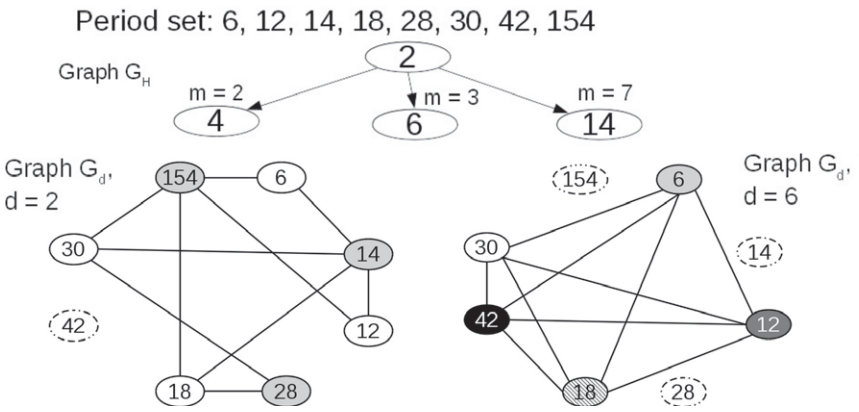


Fig. 1. A violation of the third condition of the theorem 1.

Other examples of application of theorem 1 may be found in [7].

3 Motivation and Problem Statement

In real conditions, the task period depends on technical characteristics of devices used. For example, it may correspond to the frequency of signals that are sent

or received by the task. And these characteristics are not diverse. Table 1 shows examples of widely used task periods in industrial RTOS.

So, if the number of different permissible periods is not so large, is it possible to generate all good task period sets? ("Good" means that there is a conflict-free location of the starting points.)

Table 1. Frequencies and periods. One second contains 2000 points.

Hz	Period	Hz	Period	Hz	Period
400	5	60	32-35	30	60-65
200	10	50	40	20	100
100	20	40	50	10	200

Let $p_1 < p_2 < ... < p_k$ be all possible periods. Suppose there are n_1 tasks with period p_1, n_2 tasks with period p_2, ..., n_k tasks with period p_k. The tuple $(n_1, n_2, ..., n_k)$ is called *correct* if it satisfies the condition (1) of the CPU usage. We say that the correct tuple $(n_1, n_2, ..., n_k)$ is a *solution* if there is a conflict-free location of starting points for tasks with such periods. The solutions can be partially ordered as follows. We say that $(n_1^{(1)}, ..., n_k^{(1)}) \leq (n_1^{(1)}, ..., n_k^{(1)})$ if $n_i^{(1)} \leq n_i^{(2)}$ for all $i = 1, ..., k$.

It turns out that theorem 1 provides means to construct an algorithm for generating all maximal solutions. ("Maximal" means maximal with respect to the introduced order.) Obviously, for any solution τ, there exists a maximal solution $\tilde{\tau}$, such that $\tau \leq \tilde{\tau}$. So, if we have all maximal solutions, then we can construct all solutions.

4 Algorithm Sketch

In the most general form, the generation algorithm is as follows.

Loop: divider d in G_H
  If d is root **Then**
    Loop: color distribution between periods p_i
      Color set contains d elements, which should be distributed between the periods (or vertices of the graph G_d) taking into account the conditions of theorem 1. Important: one color can refer to several periods if the opposite is not forbidden (which is possible if there is an edge between periods in the graph G_d).
    End of loop for color distribution between periods
  Else Loop: color distribution between periods p_i
      In this case it is necessary to take into account the color distributions constructed for all parents of the divider d. Colors must be inheritors

of the parent colors. If there are several parents, the inheritance must be agreed with all parents. Besides, the conditions of theorem 1 must be satisfied.

End of loop for color distribution between periods

End of loop for d

Note that all colors are distributed, so we do get the maximal solutions. In addition, the information obtained during the construction of the maximal tuple makes it possible to build immediately a conflict-free schedule for this tuple.

On the basis of the above scheme, we have developed a generator of the solution set. Our implementation generates the desired set of maximal solutions in an acceptable time. The results of the experiments are given below in section 6.

5 Application of the Generator

Now let's discuss how to use such a generator in practice.

A generated set of solutions gives an answer to the first question of scheduling: is it possible to build a conflict-free distribution of starting points for a given set of tasks. In addition, we can get a schedule for starting points.

Complete information about available solutions allows to automatically distribute tasks between processors without any risk of choosing an impossible combination of periods.

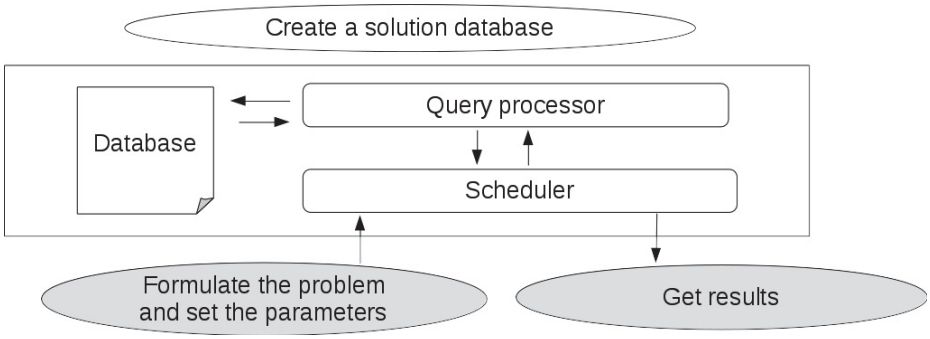


Fig. 2. A possible architecture of the automatic scheduler. Operator actions are grayed out.

An automatically scheduling may include the following components (see figure 2):

- Database which includes all generated solutions for the given period set and the schedules for these solutions.
- A query processor for standard database queries.
- Analyzer that builds a task processing schedule, in accordance with the user-specified requirements.

6 Experimental results

We conducted several experiments to generate sets of solutions for different period sets that correspond to graphs G_H with different structural complexity (see figures 3 and 4). For all produced solutions, schedules were generated and tested for compliance with the condition for conflict-free schedules from the section 1.

Table 2. The number of maximal solutions for some period sets

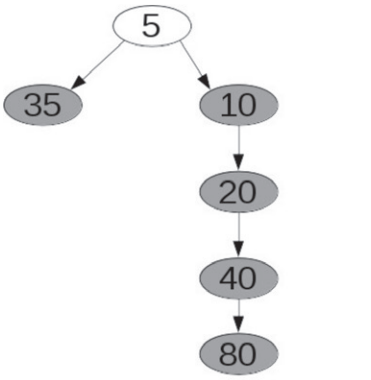
Period set	Time	The number of maximal solutions	The number of maximal correct tuples
10, 20, 35, 40, 80	0,5 sec	656	7 456
10, 20, 40, 50, 100, 200	32 min 17 sec	176 604	188 844
10,64,20,32,40, 50,100,200	42 min 34 sec	552 610	6 108 197
10,65,20,35, 40,100,200, 400, 1000	16 min 5 sec	702 264	1 263 391 852

Table 2 shows the number of maximal solutions for some period systems. The last column contains the number of maximal correct tuples for the period set. This characteristic shows that the structure of the graph G_H is more important for the number of solutions, then quantitative indicators of period set is. For example, consider two last period sets. One can see that two large periods (400 and 1000) dramatically increase the number of maximal tuples, but have little effect on the number of solutions.

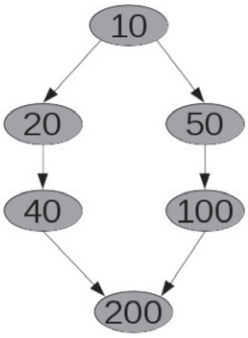
Figures 3 and 4 show that the presence of additional cross-links in G_H increases the generation time, apparently complicating generation process, while the tree structure of the graph G_H simplifies and speeds up the generation.

7 Conclusion

In this paper, we studied the problem of effective finding the starting execution points for scheduling strictly periodic tasks with given permissible periods. The

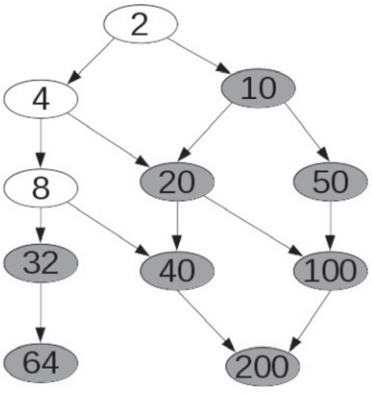


Time	Solution number
0,5 sec	656

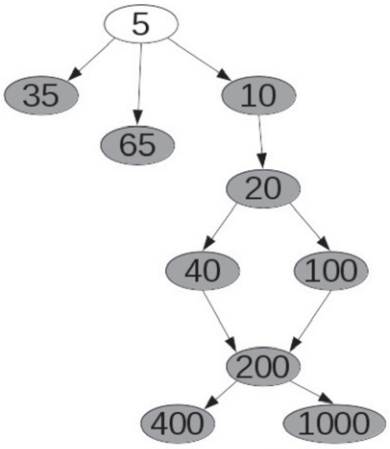


Time	Solution number
32 min 17 sec	176604

Fig. 3. Relationship of graph G_H with the number of solutions. Simple examples.



Time	Solution number
42 min 34 sec	552610



Time	Solution number
16 min 5 sec	702264

Fig. 4. Relationship of graph G_H with the number of solutions. More complicated examples.

main innovation is: instead of solving partial schedulability problem for each set of tasks, we suggest to enumerate all schedulable sets of tasks with given possible periods. Based on previously obtained theoretical results, we propose a corresponding algorithm. Our implementation of the algorithm completes in an acceptable time. The algorithm allows to build a database of schedulable sets of tasks with all data necessary for schedule construction. Then one can use this database to check any set of tasks against schedulability condition and to obtain schedule immediately.

References

1. Liu, C.L., Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. J. ACM **20**, 46–61 (1973)
2. Yomsi, P.M., Sorel, Y.: Non-schedulability conditions for off-line scheduling of real-time systems subject to precedence and strict periodicity constraints. In: Proc. of the 11th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA06. Prague (2006)
3. Yomsi, P.M., Sorel, Y.: Schedulability analysis for non necessarily harmonic real-time systems with precedence and strict periodicity constraints using the exact number of preemptions and no idle time. In: Proc. of the 4th Multidisciplinary International Scheduling Conference, MISTA09. Dublin, Ireland (2009)
4. Zelenov, S.V.: Scheduling of strictly periodic tasks in real-time systems. Trudy ISP RAN / Proc. ISP RAS **20**, 113–122 (2011) (in Russian)
5. Tretyakov, A.V.: Automation of scheduling for periodic real-time systems. Trudy ISP RAN / Proc. ISP RAS **22**, 375–400 (2012) (in Russian)
6. Zelenova S.A., Zelenov S.V.: Non-conflict scheduling criterion for strict periodic tasks. Trudy ISP RAN / Proc. ISP RAS **29** (6), 183–202 (2017) (in Russian) https://doi.org/10.15514/ISPRAS-2017-29(6)-10
7. Zelenova, S.A., Zelenov, S.V.: Schedulability Analysis for Strictly Periodic Tasks in RTOS. Programming and Computer Software **44** (3), 159–169 (2018) https://doi.org/10.1134/S0361768818030076

Научное издание

A.P. Ershov Informatics Conference

PSI Conference Series, 12th Edition

July 2–5, 2019, Novosibirsk, Akademgorodok, Russia

Preliminary Proceedings

N. Bjørner, I. Virbitskaite, A. Voronkov, Eds.

Статьи приводятся в авторской редакции

Дизайн обложки Т. М. Бульонковой
Ответственная за выпуск Н. А. Черемных
Подготовка к печати Г. Р. Семенихиной,
С. В. Исаковой, Т. А. Марковой, Е. А. Машковой, Е. В. Неклюдовой

Подписано в печать 07.06.2019 г. Формат 60x84 1/8.
Уч.-изд. л. 45,75. Усл. печ. л. 42,5.
Тираж 120 экз. Заказ № 155

Издательско-полиграфический центр НГУ
630090, г. Новосибирск, ул. Пирогова, 2

Organized by:

A. P. Ershov Institute of Informatics Systems

Novosibirsk State University
*THE REAL SCIENCE

Sponsored by:

Novosibirsk State University
*THE REAL SCIENCE

РОССИЙСКИЙ ФОНД ФУНДАМЕНТАЛЬНЫХ ИССЛЕДОВАНИЙ
РФФИ

Microsoft Research

ARQA TECHNOLOGIES

Sibers®

STI · INTERNATIONAL

STI · INNSBRUCK

ISBN 978-5-4437-0911-6