ИСП
Институт Системного Программирования
им. В.П. Иванникова
Российской Академии наук

ISSN 2079-8156 (Print)
ISSN 2220-6426 (Online)

Труды Института Системного Программирования РАН

Proceedings of the Institute for System Programming of the RAS

Том 30, выпуск 4

Volume 30, issue 4

Москва 2018

ISSN 2079-8156 (Print), ISSN 2220-6426 (Online)

Труды Института системного программирования РАН

Proceedings of the Institute for System Programming of the RAS

Труды ИСП РАН – это издание с двойной анонимной системой рецензирования, публикующее научные статьи, относящиеся ко всем областях системного программирования, технологий программирования и вычислительной техники. Целью издания является формирование научно-информационной среды в этих областях путем публикации высококачественных статей в открытом доступе.

Издание предназначено для исследователей, студентов и аспирантов, а также практиков. Оно охватывает широкий спектр тем, включая, в частности, следующие:

- операционные системы;
- компиляторные технологии;
- базы данных и информационные системы;
- параллельные и распределенные системы;
- автоматизированная разработка программ;
- верификация, валидация и тестирование;
- статический и динамический анализ;
- защита и обеспечение безопасности ПО;
- компьютерные алгоритмы;
- искусственный интеллект.

Журнал издается по одному тому в год, шесть выпусков в каждом томе.

Поддерживается открытый доступ к содержанию издания, обеспечивая доступность результатов исследований для общественности и поддерживая глобальный обмен знаниями.

Труды ИСП РАН реферируются и/или индексируются в:

Proceedings of ISP RAS are a double-blind peer-reviewed journal publishing scientific articles in the areas of system programming, software engineering, and computer science. The journal's goal is to develop a respected network of knowledge in the mentioned above areas by publishing high quality articles on open access.

The journal is intended for researchers, students, and practitioners. It covers a wide variety of topics including (but not limited to):

- Operating Systems.
- Compiler Technology.
- Databases and Information Systems.
- Parallel and Distributed Systems.
- Software Engineering.
- Software Modeling and Design Tools.
- Verification, Validation, and Testing.
- Static and Dynamic Analysis.
- Software Safety and Security.
- Computer Algorithms.
- Artificial Intelligence.

The journal is published one volume per year, six issues in each volume.

Open access to the journal content allows to provide public access to the research results and to support global exchange of knowledge. **Proceedings of ISP RAS** is abstracted and/or indexed in:

Google Scholar, ULRICHSWEB Global Serials Directory, CYBERLENINKA, OCLC WorldCat, BASE Bielefeld Academic Search Engine, OpenDOAR, eLIBRARY.RU, ROAR Registry of Open Access Repositories, Math-Net.Ru, crossref

УДК004.45

Адрес: 109004, г. Москва, ул. А. Солженицына, дом 25.
Телефон: +7(495) 912-44-25
E-mail: info-isp@ispras.ru
Сайт: http://www.ispras.ru/proceedings/

Address: 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Tel: +7(495) 912-44-25
E-mail: info-isp@ispras.ru
Web: http://www.ispras.ru/en/proceedings/

Содержание

Table of Contents

Tolerant parsing with a special kind of «Any» symbol: the algorithm and practical application

A.V. Goloveshkin <alexeyvale@gmail.com>
S.S. Mikhalkovich <miks@sfedu.ru>
I.I. Vorovich Institute for Mathematics, Mechanics and Computer Science,
Southern Federal University,
8a, Milchakova st., Rostov-on-Don, 344090, Russia

Abstract. Tolerant parsing is a form of syntax analysis aimed at capturing the structure of certain points of interest presented in a source code. While these points should be well-described in the corresponding language grammar, other parts of the program are allowed to be not presented in the grammar or to be described coarse-grained, thereby parser remains tolerant to the possible inconsistencies in the irrelevant area. Island grammars are one of the basic tolerant parsing techniques. "Island" is used as the relevant code alias, while the irrelevant code is called "water". In the paper, a modified LL(1) parsing algorithm with built-in "Any" symbol processing is described. The "Any" symbol matches implicitly defined token sequences. The use of the algorithm for island grammars allows one to reduce irrelevant code description as well as to simplify patterns for relevant code matching. Our "Any" implementation is more accurate and less restrictive in comparison with the closest analogues implemented in Coco/R and LightParse parser generators. It also has potentially lower overhead than the "bounded seas" concept implemented in PetitParser. As shown in the experimental section, the tolerant parser generated by the C# island grammar is proven to be applicable for large-scale software projects analysis.

Keywords: tolerant parsing; robust parsing; lightweight parsing; partial parsing; island grammar; parser generation

DOI: 10.15514/ISPRAS-2018-30(4)-1

For citation: Goloveshkin A.V., Mikhalkovich S.S. Tolerant parsing with a special kind of "Any" symbol: the algorithm and practical application. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 4, 2018. pp. 7-28. DOI: 10.15514/ISPRAS-2018-30(4)-1

1. Introduction

Tolerant parsing is a parsing technique differing from the detailed whole-language (so-called baseline) parsing needed to build a full-featured compiler for a certain programming language. The main feature of the approach is the ability to capture points of interest inside the program, while all the code that does not contain such points can be skipped with no or minimal analysis performed. From developer's perspective, this feature allows her to focus on the structure of the points of interest, providing a minimal description of the irrelevant area. Tolerant parsing is usually called *lightweight* because tolerant grammar tends to be much shorter than the baseline one.

There are several reasons for the tolerant parsing to be the most suitable option for the program analysis

- **Language embedding**: Some program artifacts assume the usage of multiple languages in one source file. In yacc-like grammars describing the syntax-directed translation, actions performed on a parsing step are expressed in terms of a certain general-purpose language. This means that the parser developed to capture the grammar structure must be tolerant to all the possible variations of these language snippets. A possible application of a tolerant grammar parser is described in [1]. A detailed description of the embedded language tolerant parsing is given in [2].
- **Full grammar inaccessibility**: Tolerant grammar imprints the developer’s notion of what places inside the program are the most important in the context of the current task. Its structure and the mapping between the grammar entities and the language constructs are transparent to the programmer from the very beginning and can be further refined in accordance with the in-the-wild testing results. On the contrary, the baseline grammar usage requires a prior exploration and comprehension. This process is proved to be time-consuming [3] and can be impossible due to proprietary issues or manual baseline parser writing [4].
- **Domain-specific idioms**: In a certain project, some local domain-specific patterns can be applied [4]. They represent a high-level abstraction layer which is not presented in the language syntax and obviously is out of scope of the whole-language parser. Nevertheless, tolerant parsers can be strictly focused at these patterns, ignoring the underlying structure that allows one, in particular, to perform the impact analysis [5].
- **Incorrect program processing**: Syntax errors can be handled by the whole-language parser with some sophisticated error recovery mechanisms [6, 7]. These mechanisms are heuristic by the nature and do not guarantee the successful parsing resumption, as well as the preservation of the built parts of the parse tree. Tolerant parser is able to skip irrelevant error-containing areas. At the same time, tolerant parsing can be broken by the mismatch of the elements structuring the program (e.g. by the absence of a block closing bracket in C#). Specific error handling techniques allowing recovering from this category of errors are described for the *bridge grammars* [8, 9], a special kind of the island grammars.

The contributions of this paper are: 1) a modification of the standard LL(1) parsing algorithm aimed at *island grammars* tolerant parsing paradigm and designed to simplify irrelevant code skipping by means of a special `Any` symbol, this symbol is used in a tolerant grammar to mark an irrelevant code without specifying its structure; 2) a compiler generator with a built-in tolerant grammar description language containing `Any` as a part of the standard syntax; 3) a lightweight grammar of the C# programming language for this generator; 4) an experimental evidence of the applicability of the generated tolerant C# parser for large-scale software projects analysis.

The remainder of the paper is organized as follows: a brief overview of the existing tolerant parsing techniques is provided in Section 2, in Section 3 the main goals of the current research are listed, in Section 4 we discuss related work and outline limitations of the closest analogues of our approach, in Section 5 the modification of the standard LL(1) parsing algorithm aimed at `Any` symbol processing is introduced. The tolerant grammar for the C# programming language is presented in Section 6, this section also includes a sufficient volume of experimental data obtained by applying the generated tolerant parser to a real-world software source code. In Section 7 a summary of the theoretical and practical contribution of the paper is provided.

2. Tolerant parsing techniques

Three basic tolerant parsing techniques considered in [2, 4, 5, 10–13] are fuzzy parsing, island grammars and skeleton grammars.

Fuzzy parsing is based on the notion of *anchors,* specific tokens that mark the beginning of the constructs of interest. The formal definition of a fuzzy parser is provided in [10, 11]. The grammar used by the fuzzy parser actually consists of a number of smaller grammars. Each of them has its own start symbol with a production rule starting with the anchor. The main concern of the fuzzy parsing technique is that parsing process is tightly coupled with anchor tokens and can be error-prone in case these tokens appear outside of the points of interest.

Skeleton grammar construction is described in [12]. The skeleton grammar partially shares its structure with the baseline grammar. Rules describing points of interest are complemented with baseline grammar rules needed to derive those points from the start symbol (this process is called *root completion*). After the root completion, special *default productions* are formulated for all the undefined nonterminal symbols appearing in the rules added. The key precondition making this process possible is the baseline grammar accessibility. As noticed in Section 1, most often this is not the case, besides, baseline grammar comprehension is quite time-consuming and requires some additional effort.

Island grammars technique is in the focus of our research. We believe that the concept of an island grammar is not well-known, so we provide its formal definition in accordance to [4, 5], despite the fact that this definition is not further referenced.

Definition 1. Given a context-free grammar $G = (N, T, P, S)$, where N is a set of nonterminal symbols, T is a set of terminal symbols, P is a set of production rules, $S \in N$ is a specified start symbol, and a set of constructs of interest $I \subset T^*$ such that

$\forall i \in I, \exists \omega_1, \omega_2 \in T^*: \omega_1 i \omega_2 \in L(G)$, where $L(G)$ denotes the language generated by G. An *island grammar* $G_I = (N', T', P', S')$ for $L(G)$ has the following properties:

1) $L(G) \subset L(G_I)$;
2) $\forall i \in I, \exists n \in N': n \overset{*}{\Rightarrow} i$ and $\exists \omega_1, \omega_2 \in T^*: \omega_1 i \omega_2 \notin L(G) \wedge \omega_1 i \omega_2 \in L(G_I)$;
3) $K(G) > K(G_I)$.

The first property means that G_I generates an extension of $L(G)$, the second means that the syntax analyzer for G_I recognizes constructs of interest from I in at least one sentence that is not recognized by the parser for G. The third property introduces the function $K(G)$ denoting the grammar complexity.

Informally speaking, island grammar consists of detailed productions describing certain constructs of interest (the *islands*) and liberal productions that catch the remainder (the *water*). Island productions form a set of *patterns* to be matched by the points of interest. However, patterns are not enough to overcome two important island grammars side effects called *false positives* and *false negatives* [12]. In case relevant code snippets look similar to the irrelevant ones, they can be confused by the parser, as a result, the irrelevant code will be recognized as the point of interest and some points of interest will be missed, there also can be a parse error. To minimize the mismatch, iterative refinement is needed for patterns as well as for *anti-patterns* matching irrelevant code.

To reduce the need for anti-patterns description and refinement, indeterministic parsing techniques are usually used. GLR [14, pp. 381–391] and GLL [15] parsers are capable to apply multiple parse actions for the same token in case of an ambiguity and continue parsing the program in all ways. However, they have a number of disadvantages: indeterministic parsing is hard to trace and debug, may return multiple parse trees that need some extra processing, and in case the islands look similar to the water, a parsing result can be extremely unpredictable. From the latter it follows that one still has to describe and refine some anti-patterns.

3. Problem statement

The key assumption of the current research is that tolerant parsing can be performed with a deterministic algorithm, while patterns and anti-patterns forming the tolerant grammar can be simplified and partially eliminated by making the algorithm capable to match and skip some token sequences which have no explicit definition in the grammar.

The key goals of the current research are:

1) to design an LL(1) parsing algorithm with built-in notion of a special `Any` grammar symbol that provides skipping of the token sequences that are not explicitly described in the grammar;

2) to develop a compiler generator with an integrated language for LL(1) grammars writing, supporting `Any` symbol usage and automatic syntax tree construction;
3) to implement a tolerant island grammar for the C# programming language in the format supported by the generator below; the grammar is supposed to contain water anti-patterns simplified with `Any` symbol;
4) to test parser's applicability to the analysis of large-scale software projects.

The developed tool is planned to be used for lightweight parsing of software projects and their further sustainable concern-based markup.

4. Related work

4.1 Coco/R

The first tool with embedded capability to match tokens from sets which are not directly specified in a grammar is the Coco/R recursive-descent parsers generator. According to the documentation [16, p. 14], a special symbol `ANY`, which denotes any token that is not an alternative to that `ANY` symbol in the current production, is predefined in generated parsers. For a given grammar, an individual set of admissible tokens is connected with each `ANY` entry. Initially all the sets consist of all the tokens defined in the grammar, then at the parser generation stage the alternatives of `ANY` symbols are removed from the corresponding sets to make the situation when a parser has to make a choice between `ANY` and some explicitly specified token unambiguously solvable in favor of the explicit option. Further we will call these alternatives *rivals*, in order to avoid terminological confusion with alternatives forming grammar rules.

The major shortcoming of `ANY` implementation in Coco/R is that the intuitive principle of the explicitly specified token priority is both incomplete and excessively restrictive. As a result, there are grammars for which parsers generated by Coco/R do not parse some programs valid from the developer's point of view. Some examples of such Coco/R grammars are shown in fig. 1. Lower case is used for terminal symbols, `{}` denotes zero or more repetitions of bracketed elements.

Excessive restrictiveness manifests itself for the iteration `{ANY}`, for which the same set defines admissible tokens both for the first position in the sequence corresponding to `{ANY}` and for the rest positions. For the grammar in Fig. 1a, the set `{b, c}` corresponds to `ANY`. The token `d` is excluded from the set to make parser capable to finish `{ANY}` matching and match `d` explicitly. The token `a` is also excluded, that makes all the strings starting from `a` being matched by the first alternative with explicitly written `a` token in the beginning. However, the latter exclusion leads to the fact that the string `bad$` is not recognized by the parser. Note that the first token of the input — token `b` — is enough to choose the right production for `A` nonterminal, and the next `a` token cannot be treated as the beginning of the first alternative. If separate sets were used for `ANY`s from the iteration, `a` could be added to the set of admissible tokens for the second and subsequent positions in `{ANY}`, and this would not lead to an ambiguity.

The lack of outer context analysis for nonterminal symbols leads to incompleteness of the constraints that are imposed on the `ANY` admissible tokens set. In Fig. 1b, `ANY` has no rivals within the rule, so the set of admissible elements consists of all the tokens defined in the grammar. As a result, the generated parser is not capable to recognize the intuitively recognizable string `abcd$`. Once `{ANY}` processing starts, the parser reaches the end of the input stream treating each token as a part of the sequence corresponding to `{ANY}`. Outer context analysis for nonterminal `B` shows that token `d` is in $\mathrm{FOLLOW}(B)$, so it may appear after `ANY` iteration. Hence `d` must be deleted from admissible tokens set and be matched explicitly.

```
A = a b c | {ANY} d.      A = B d | b B.  B = a {ANY}.
          (a)                            (b)
```

Fig. 1. The grammars illustrating ANY implementation shortcomings in Coco/R.

At the same time, static analysis of the outer context is too coarse to be a good solution. For the grammar in Fig. 1b, `B` appears in two different contexts and the restriction derived from the first context (`B d` alternative) is not relevant for the second one (`b B` alternative). In the second context, `ANY` has no inner or outer rivals. With statically defined admissible tokens set, two contexts are mixed and string `bad$` is not recognizable again. On the other hand, after choosing an alternative for `A`, the more precise information about what can follow `{ANY}` is available. That is, with dynamic decision making at the parsing stage, the set of programs recognizable by the parser can be extended.

In the current paper, the symbol `Any` is described. Unlike the `ANY` symbol in Coco/R, it corresponds to the sequence of zero or more tokens, not a single token. In its implementation, all the shortcomings listed above are eliminated and the decision about the current token's admissibility at `Any` position is made at the parsing stage.

4.2 LightParse

The tool for lightweight LALR(1) parsers development called LightParse [17] also supports the use of Coco/R-like `Any` symbol. LightParse application is similar to what we plan to do: generated lightweight parsers are used for concern-oriented source code markup [18]. LightParse performs static construction of the sets of tokens allowed at `Any` position and inherits all the Coco/R `ANY` implementation limitations. Besides, LightParse grammar is not directly used to generate the parser. Instead, it is translated to the YACC-like format supported by the standard LALR(1) parser generator GPPG, then GPPG produces the parser. In the translated grammar, every entry of `Any` symbol is presented as a nonterminal symbol with single-element alternatives, by an alternative for each of the admissible terminal symbols.

To get the valid YACC-like grammar without the nonterminal outer context analysis, LightParse imposes additional restrictions on `Any` usage: this symbol is not permitted to be in the end of the alternative, except for the start symbol productions. The presence of the intermediate grammar processing stage leads to inconsistency between the source grammar vocabulary, which is used by the grammar developer too, and the terms used in messages issued by the GPPG generator when some parser generation errors appear. Our `Any` implementation does not assume additional grammar adaptations for making the grammar suitable for the standard parsing algorithm. Instead, the standard LL(1) algorithm is modified to integrate the notion of `Any` and make it possible to define admissible tokens dynamically at the parsing stage. This eliminates the limitations of LightParse `Any` symbol.

4.3 Bounded seas

In [19], an extension of the regular parsing algorithm is described for parsing expression grammars (PEG). It is intended to automatically deduce anti-patterns for water which is supposed to be context-aware, i.e., specific for each particular island in the input. This approach named *bounded seas* is integrated in PetitParser tool which allows one to implement PEG-based parsers. Bounded seas are intended to completely eliminate the need for water rules explicit description in island grammars. A rule element of the form `~island~` is treated as a triple `before-water island after-water`. The key property of the water is that it never consumes any input from the right context of the bounded sea. The right context can be derived statically from the grammar or dynamically from the parser state. For the `after-water` entity, right context is set with an expression consisting of all the possible expressions that can directly follow `after-water`, separated with the ordered choice operator. Right context for the `before-water` consists of the `island` expression itself and the corresponding `after-water` boundary expression which both are ordered choice operands. Water expression succeeds when the corresponding right boundary expression succeeds.

Checking all the possible boundaries assumes backtracking, which leads to a sufficient time overhead. Since backtracking is a basic technique for PEG due to ordered choice operator presence, it is usually optimized with *packrat* parsing [20], which makes parsing time linearly dependent on the length of the program. Similar technique is used to eliminate potentially exponential complexity of bounded seas analysis. However, execution time decrease is achieved at the cost of a significant increase in the amount of memory used. Despite the right context exploration complexity, bounded seas are not able to make a globally correct decision on when water skipping should be ended. It is outlined in [19] that expressions forming the sea boundary actually recognize only prefixes of the possible boundaries, and boundaries form an LL(k) language where k depends on the particular situation. So, being designed to eliminate the need for anti-patterns presence in a grammar, bounded seas, however, do not guarantee successful distinguishing islands from water without any explicit hints about the water content.

Besides, PetitParser itself is Smalltalk-based[1] and is intended for use in a closed ecosystem of Pharo virtual machine and Moose framework, so generated parsers have extremely low integrability into an arbitrary project.

The approach presented in the current paper has less overhead because it does not use backtracking at all. It performs a linear input processing and use the modified `FIRST` set building algorithm to find a boundary for `Any`. Though in [19] standard `FIRST` and `FOLLOW` sets from LL(1) parsing theory are named insufficient to recognize the boundary, it is demonstrated in Section 6 that with proper formulation of anti-patterns, the use of a modified `FIRST` set is enough to successfully analyze large-scale software project sources. `Any` symbol is used instead of explicit description of some parts of patterns and anti-patterns, that makes the island grammar significantly shorter and simplifies the grammar development process. Our parser generator is implemented in C#, thus it can be used with projects written on any .NET Framework-supporting language.

5. «Any» symbol implementation

We are mainly focused not on the individual islands capturing but on the extraction of the program hierarchical structure up to a certain level and tend to name the relevant code not *islands* but *land*, so the developed parser generator was named LanD[2] (by coincidence, it is also an acronym of «language description»). Table-driven predictive LL(1) parsing algorithm [21, pp. 220–228] was selected as the simplest and most suitable for debugging option for water skipping integration.

5.1 Formal definition of a simplified grammar

We introduce into the grammar the special terminal symbol `Any` to mark places where zero or more tokens from the irrelevant area can be matched. We denote by $\mathrm{lhs}(p)$ and $\mathrm{rhs}(p)$, respectively, the left and the right part of the production p. Notation $x \in \mathrm{rhs}(p)$ for $x \in N \cup T$ means that $\mathrm{rhs}(p) = \alpha_1 x \alpha_2$, where $\alpha_1 \in (N \cup T)^*, \alpha_2 \in (N \cup T)^*$. $\mathrm{SYMBOLS}(\gamma)$ is used for the set of terminal symbols needed to compose all the $\omega: \gamma \overset{*}{\Rightarrow} \omega, \gamma \in (N \cup T)^*, \omega \in T^*$. Through the symbol `Any`, we formulate the concept of a simplified grammar.

Definition 2. Let $G = (N, T, P, S)$ be a context-free grammar, $Any \notin T$. *Simplified* with respect to G is the grammar $G_s = (N_s, T_s, P_s, S_s)$ defined as follows:

1) $S_s = S$;

[1] PetitParser was also ported to a number of other languages, but those ports are experimental and are not updated with state-of-the-art features such as bounded seas.

[2] https://github.com/alexeyvale/SYRCoSE-2018

2) $P_s = \{p \in f(P) \mid \text{lhs}(p) = S_s \vee \exists p' \in P_s: \text{lhs}(p) \in \text{rhs}(p')\}$, where $f: P \to \{p = A \to \alpha \mid A \in N, \alpha \in (N \cup T \cup \{Any\})^*\}$ is the mapping that satisfies the following criteria:
   a) $\exists P' \subseteq P: P' = \{p \in P \mid f(p) \neq p\}$, $P' \neq \emptyset$,
   b) $\forall p \in P \setminus P'$, $f(p) = p$,
   c) $\forall p \in P'$, $\exists n \in \mathbb{N}$: p is representable in the form $A \to \alpha_1\gamma_1\beta_1\alpha_2\gamma_2\beta_2 \ldots \alpha_n\gamma_n\beta_n$ and $f(p)$ is representable in the form $A \to \alpha_1 Any \beta_1 \alpha_2 Any \beta_2 \ldots \alpha_n Any \beta_n$, where $\forall i \in [1..n]$, $\alpha_i\gamma_i\beta_i \in (N \cup T)^*$, and $\forall i \in [1..n]$, $\forall a \in \text{FOLLOW}(A)$, $\text{SYMBOLS}(\gamma_i) \cap \text{FIRST}(\beta_i\alpha_{i+1}\gamma_{i+1}\beta_{i+1} \ldots \alpha_n\gamma_n\beta_n a) = \emptyset$;
3) $N_s = \{A \in N \mid \exists p \in P_s: \text{lhs}(p) = A\}$;
4) $T_s = \{a \in T \mid \exists p \in P_s: a \in \text{rhs}(p)\} \cup \{Any\}$.

Intuitively, P_s contains productions for the start symbol of G_s and productions for all the nonterminals which are reachable from the start symbol. Note that, according to items 3 and 4, $\forall p \in P_s$, $\text{lhs}(p) \in N_s, \text{rhs}(p) \in (N_s \cup T_s)^*$, i.e. P_s really satisfies the production set definition for a context-free grammar.

The definition of the mapping f means that some of the strings generated by G contain substrings which can be replaced with `Any`, then we obtain strings generated by G_s. In the absence of grammar simplification options developer has to work with grammar G, which can correspond to the baseline language grammar, as well as be a specially written more tolerant version of the baseline grammar, containing all the anti-patterns described explicitly. If `Any` symbol is supported by the grammar and the corresponding generator, anti-patterns forming a set P' can be substantially simplified. Symbol `Any` can be written instead of the parts denoted by γ_i in production's right hand side in case these parts satisfy the criterion 2c of the definition 2. Verification of this criterion is possible only when solving a direct problem: when the grammar G_s is generated based on the existing G. In a real situation, there is no grammar G and the developer has to solve the inverse problem: she manually writes a simplified grammar G_s, assuming that her knowledge of the particular island patterns and the general structure of the program is close to the ground truth — the structure of the baseline grammar G — and also considering parts denoted by `Any` satisfy the criterion. When this is not the case, unparsed or incorrectly parsed programs appear at the testing phase, this means that the grammar should be refined. This process usually takes several iterations.

Notice that despite the parser is built according to grammar G_s, a program from $L(G)$ is needed to be parsed. The modified LL(1) algorithm uses the criterion 2c to translate the program to the language $L(G_s)$.

5.2 Parsing algorithm modification

In fig. 2a the modified LL(1) parsing algorithm is presented. The highlighted lines distinguish it from the standard algorithm. In the given pseudo-code parsing stack is accessed through the `Stack` variable, input buffer is accessed through the lexical analyzer object `Lexer` with methods `NextToken` returning the next token from the input stream and `CurrentToken` returning the last token that was read. The variable `t` corresponds to an additional buffer for the current token, `M` denotes the parsing table. The grammar G_s is a regular LL(1) grammar where `Any` is a regular token, therefore parsing table construction algorithm remains unmodified and the construction itself is carried out in the standard way.

```
Stack.Push($);
Stack.Push(S);

X := Stack.Peek();
t := Lexer.NextToken();
while (X ≠ $) do
  if (X = t) then
    if (t = Any) then                                   (1)
      Stack.Pop();
      t := Lexer.CurrentToken();
      while (t ∉ FIRST'(Stack) and t ≠ $) do
        t := Lexer.NextToken();
      end while;
      if (t = $ and $ ∉ FIRST'(Stack)) then
        error();
      end if;
    else                                                (2)
      Stack.Pop();
      t := Lexer.NextToken();
    end if;
  elif (M[X,t] = X → Y1Y2...Yk) then                    (3)
    Stack.Pop();
    for (i from k to 1) do
      Stack.Push(Yi);
    end for;
  elif (t = Any) then                                   (4)
    error();
  else                                                  (5)
    t := Any;
  end if;
  X := Stack.Peek();
end while;

if (t = $) then                                         (6)
  accept();
else
  error();
end if;
```

(a)

```
BuildFirst'():
  foreach (A ∈ N) do
    MemorizedFirst'[A] := Ø
  end foreach;
  changed := true;
  while (changed) do
    changed := false;
    foreach (A → α ∈ P) do
      MemorizedFirst'[A] ∪= FIRST'(α);
      if (MemorizedFirst'[A] is changed) then
        changed := true;
      end if;
    end foreach;
  end while;
```

(b)

```
FIRST'(α = Y1Y2...Yk):
  first := Ø;
  for (i from 1 to k) do
    if (Yi ∈ T\{Any}) then
      first ∪= {Yi};
      break;
    elif (Yi ∈ N) then
      first ∪= MemorizedFirst'[Yi]\{ε};
      if (ε ∉ MemorizedFirst'[Yi]) then
        break;
      end if;
    end if;
  end for;
  if (∀i ∈ [1..k]: ε ∈ MemorizedFirst'[Yi]
      or Yi = Any) then
    first ∪= {ε};
  end if;
  return first;
```

(c)

Fig. 2. Modified algorithms: (a) LL(1) parsing algorithm, (b) FIRST set memorization algorithm, (c) FIRST set building algorithm

Modification of the parsing algorithm is caused by the fact that parser do a more complicated job than checking if the program is valid with respect to G_s. While parser is generated by the simplified with respect to some G grammar G_s, the program derived by G comes as the input. As tokens are received from the input stream, the modified parser should translate the program from $L(G)$ to $L(G_s)$, then it can check the syntactic correctness of the translated part.

When the terminal symbol on the top of the parsing stack does not match the current token in `t` or when a nonterminal symbol `X` is on top of the stack and there is no record in the cell `M[X, t]`, the standard LL(1) algorithm reports an error because there is no explicit option available to continue parsing, and possibly starts an error recovery routine. For the modified algorithm, this situation is normal because, as it was said, the program does not belong to the language the parser is generated for. In case `Any` is on the top of the stack or the `M[X, Any]` cell is not empty, the modified algorithm tries to replace with `Any` some sequence of tokens from the input stream, making the transition from the text from $L(G)$ to the text from $L(G_s)$. Replacement is based on the criterion 2c: the set of tokens forming the replaced sequence must not intersect with the set of tokens which are possible `Any` successors in accordance with the parsing stack state. The successors set is called `FIRST'`, it is built by the modified version of the standard `FIRST` algorithm. This modification is discussed in Section 5.3. Obviously, $L(G) \subseteq L(G_s)$, because at the `Any` position not only valid $L(G)$ program subsequence can be replaced, but also an arbitrary sequence of tokens from the complement of a successors set. This makes parser less sensitive to possible errors in water regions.

It is possible to draw some parallels between the modification given and well-known error recovery algorithms [21, pp. 228–231, pp. 295–297]: `Any` symbol looks similar to the `error` token denoting place in the grammar where recovered parsing can be resumed, `FIRST'` set seems like the set of synchronization tokens. There are grounds for such an analogy. The program parsed is erroneous in terms of G_s. Replacing tokens with `Any`, the parser looks for a place from which the program satisfies the grammar again. However, behind a skin-deep similarity, there is a fundamental difference in goals, implementation and results obtaining by the algorithms. Standard error recovery is performed when a program processed is clearly incorrect. The main goal of the recovery is to resume parsing at any cost. Some significant results of the previous analysis can be discarded, and a significant part of the input stream, possibly containing some points of interest, is discarded too. In addition, recovery is not guaranteed to be successful. According to Section 5.1, the goal of `Any` processing is the translation of a presumably valid $L(G)$ program to $L(G_s)$. The premise that the program under consideration is correct with respect to G in conjunction with the observance of the criterion 2c makes input tokens discarding totally predictable. One can be sure that the parts of the input stream replaced with `Any` belongs to the water and can be skipped without loss of the land. Furthermore, as it was previously noted, predictable and correct replacement with `Any` is possible in some cases even for programs that are incorrect with respect to G.

Further, speaking of the fact that `Any` successfully replaces a sequence of tokens of the input program, we will simply say in some cases that `Any` *matches* this sequence. Keep in mind, that, as shown below, this process is more complex than the standard token matching.

5.3 The problem of consecutive «Any»

To get tokens denoting the end of the sequence that corresponds to `Any`, the first intention is to build the standard `FIRST` set for a parsing stack, treating the symbols on the stack as a string starting from its top. Unfortunately, there is a case when the standard `FIRST` algorithm is not enough. Sometimes two or more `Any` tokens can follow each other at the beginning of the sentences which can be derived from the stack. For a grammar

```
A = Any B C; B = a | ; C = Any c;
```

the `FIRST(Stack)` set built when the first `Any` is processed equals to `{a, Any}`. The `Any` token is never returned by the lexical analyzer, so, there is no chance that parser will recognize a string with no `a` tokens. As a result, a part of $L(G)$ remains uncovered by the parser, and the valid with respect to G_s program `Any Any c` will never be recognized, because there is no input program that can be transformed to it. For the example input `bbbc$`, `Any` processing starts at the first `b` and fails at the endmarker symbol `$`.

To make the parser capable to cope with a simplified grammar that allows consequent `Any` symbols in some derivations, it is needed to modify the standard `FIRST` algorithm on the basis of the definition 2. According to it, `Any` denotes the place where the matched sequence from an input program may be empty. In the example above, the `c` terminal which is explicitly presented in the grammar can be treated as the end of the sequence to replace, if we assume that the sequence matched by the second `Any` is empty. Acting under this assumption, the modified algorithm should expand the `FIRST` set with the tokens that may follow the last of the subsequent `Any` symbols. This turns the standard `FIRST` set into the `FIRST′` used in Fig. 2a.

In fig. 2b and fig. 2c, modified algorithms for `FIRST′` construction are presented. In fig. 2b, there is an adopted version of the algorithm from [14, pp. 239–240]. It performs non-recursive construction of `FIRST′` sets for all the nonterminals in the grammar. The sets constructed are memorized in the `MemorizedFirst′` dictionary. The original algorithm is proven to be finite, the same proof is valid for the adopted version. `FIRST′` itself is presented in Fig. 2c. Note that `Any` is not placed in the `FIRST′` set.

As shown in Section 6.1, when to match a sequence of `Any` is the only available option for processing some part of the input, `FIRST′` helps to find the actual input subsequence corresponding to the whole sequence of `Any` symbols. Technically, in this case input subsequence is matched by the first `Any`, the following `Any` symbols match empty sequences. This is the only possible solution for a simplified grammar, because to say for sure how to precisely establish a pairwise match between the parts of the input subsequence and consecutive `Any` symbols, we need more information about the original G grammar. A similar problem called *overlapping seas* is discussed in [19]: when one sea may follow another, it is impossible to distinguish between the `after-water` of the first sea and the `before-water` of the second, so the second water is believed to be empty.

The suggested `FIRST′` modification is proven to be enough to develop a working tolerant grammar for the real programming language.

6. Experiments

6.1 Model example

Consider the following grammar:

```
A = B Any C; B = a | Any c; C = Any b | c;
```

The corresponding parsing table is presented in Table 1. The rows correspond to the nonterminal symbols defined in the grammar, the columns correspond to the tokens that may appear in the buffer `t`. Each cell contains the alternative that should be applied when the row nonterminal is on the top of the parsing stack and the column terminal is the lookahead token. The work of the modified parsing algorithm for a given input string is described in Table 2. Each row corresponds to the iteration of the outer `while` cycle in Fig. 2a, the last row corresponds to the action that takes place right after exiting the cycle. The numbers in the **Action** column correspond to the conditions numbered in Fig. 2a, the number of the true condition for the current iteration is placed in the table cell.

Table 1. Parsing table for the model example

	a	**b**	**c**	**Any**	**$**
A	$A \to B\ Any\ C$			$A \to B\ Any\ C$	
B	$B \to a$			$B \to Any\ c$	
C			$C \to c$	$C \to Any\ b$	

This example illustrates some of the advantages of our `Any` implementation, that were declared earlier. In contrast to the situation discussed for the Coco/R parser generator and the grammar in Fig. 1a, at the 4th iteration, the first `a` token in the input is included in the sequence being matched by `Any`, because the `Any` symbol is really rivalled by `a` only at the 1st iteration where the choice between $B \to a$ and $B \to Any\ c$ productions has to be made. The 7th iteration reveals the situation specified in Section 5.3: there is a derivation where two `Any` follow each other. Searching for all the tokens that may appear after `Any` in $A \to B\ Any\ C$ in accordance to the parsing stack, the `FIRST′` algorithm looks beyond the `Any`, which is in the beginning of $C \to Any\ b$, and considers `b` as the possible successor of the sequence that should be matched by the current `Any`. As mentioned earlier, in case `Any` is immediately followed by other `Any` symbols, a sequence of input tokens of the maximum possible length is replaced with the first `Any`, and subsequent `Any` symbols correspond to zero-length subsequences of the input.

Table 2. Tracing table

	Stack	Input	X	T	Action	Remark
1	$ A	bacaab$	A	b	(5)	
2	$ A	bacaab$	A	Any	(3)	
3	$ C Any B	bacaab$	B	Any	(3)	
4	$ C Any c Any	bacaab$	Any	Any	(1)	$\text{FIRST}'(c\ Any\ C) = \{c\}$
5	$ C Any c	caab$	c	c	(2)	
6	$ C Any	aab$	Any	a	(5)	
7	$ C Any	aab$	Any	Any	(1)	$\text{FIRST}'(C) = \{b, c\}$
8	$ C	b$	C	b	(5)	
9	$ C	b$	C	Any	(3)	
10	$ b Any	b$	Any	Any	(1)	$\text{FIRST}'(b) = \{b\}$
11	$ b	b$	b	b	(2)	
12	$	$	$	$	(6)	

Dynamically performed computation of the set of symbols that may follow `Any` takes into account the actual outer context for the alternatives that are matched (this context is formed by the elements that are lower on the stack, than the current alternative), rather than all the possible outer contexts which can arise according to the grammar.

6.2 Real-world repositories analysis

To test the algorithm on real source code repositories, the island grammar for the C# programming language was developed. The generated parser was applied to the repositories of three industrial projects ranked from the smallest to the largest by the number of files with a source code: the LanD project itself (93 files), PascalABC.NET[3] (2725 files), and Roslyn[4] (8027 files). PascalABC.NET is a programming language which combines Pascal syntax with .NET framework functionality. The corresponding project consists of compiler and IDE sources. Roslyn is a pair of open-source compilers for C# and Visual Basic. Roslyn project includes compiler sources and lots of test files capturing different complex and uncommon variants of a C# program. The number of files in the corresponding repositories relevant at the time of experiment conducting is given in brackets.

[3] https://github.com/pascalabcnet/pascalabcnet

[4] https://github.com/dotnet/roslyn

```
namespace_content  =  opening_directive*! (attribute|namespace|namespace_member)*
opening_directive  =  ('using'|'extern') Any ';'
namespace =  'namespace' name '{' namespace_content '}'
namespace_member   =  name? (enum|delegate|class_struct_interface)
enum     = 'enum' name Any '{' Any '}' ';'?
delegate = 'delegate' name before_body? ';'
class_struct_interface = ('class'|'interface'|'struct') name Any '{' class_content_element* '}' ';'?
class_content_element  = attribute | keyword_marked_entities
                       | name (keyword_marked_entities | class_member_tail)
keyword_marked_entities =  enum | delegate | class_struct_interface | operator | event
operator           =  'operator' Any arguments class_member_tail
event              =  'event' name class_member_tail
class_member_tail =  before_body? (block init_value? | initializer | ';')
before_body        =  Any ':' (arguments|Any)*
initializer        =  init_expression | init_value
init_expression    =  '=>' (Any|block)* ';'
init_value         =  '=' (Any|block)* ';'
name               =  (ID|arguments|'extern') name_tail_element*
name_tail_element =  ID|arguments|'extern'|'.'|'?'|'<' name_tail_element* '>'|'[' Any ']'|','|'::'

attribute  =  '[' (Any|attribute)* ']'
block      =  '{'(Any|block)* '}'
arguments  =  '(' (Any|arguments)* ')'
```

Fig. 3. Rules of the tolerant C# grammar for LanD parser generator

Rules from C# tolerant grammar are presented in Fig. 3, the complete grammar can be found in LanD project repository[5]. Water rules are highlighted. Symbol `*` denotes zero or more element repetitions, `+` denotes one or more repetitions, `?` denotes an optional element, brackets `()` are used for grouping. Quantifiers of a special kind, `*!` and `?!`, are used to set the non-empty alternative priority in case the ambiguity is detected at the parsing table construction stage. With their help, in particular, the dangling else problem is solved in the Pascal language grammar:

```
if = 'if' Any 'then' operator ('else' operator)?!
```

In the C# grammar, the `*!` construct is used to distinguish between `extern alias` declaration and the header of a method written in an unmanaged code. Though these constructs do not appear at the same nesting level in real programs, they are allowed to do so according to the lightweight grammar. This results in ambiguity that needs an additional priority indication.

As it can be seen, `Any` is widely used for denoting places which are insufficient for points of interest capturing. Such irrelevant areas are inheritance specification and type restrictions in class definitions (`before_block` nonterminal), field and property initializers (`initializer` nonterminal and nonterminals which are directly derivable from it). The largest parts that are matched by `Any` are blocks of code in method bodies (`block` nonterminal). A detailed description of these areas would make the grammar several times longer. In the corresponding anti-pattern formulated with `Any`, only a minimal structuring information should be placed: boundary tokens `{` and `}` are specified and self-nesting is explicitly allowed to ensure that boundaries will be matched pairwise. This technique is also used for

[5] https://github.com/alexeyvale/SYRCoSE-2018/blob/master/LanD Specifications/sharp.land

`attribute` and `arguments` entities, so it can be said that it forms a sustainable grammar writing pattern.

`Any` also appears in some patterns, such as `enum`, `class_struct_interface`, `operator`, denoting lakes among the land. Lakes can mark irrelevant places as well as places for which we are interested only in the list of matched tokens, not in the correct subtree specifying the deeper structure.

Table 3. Numbers of unparsed files per C# grammar refinement iteration.

	LanD	**PascalABC.NET**	**Roslyn**
0	8	-	-
1	0	39	-
2	0	0	209
3	0	0	31
4	0	0	3

In Table 3, the quantitative data describing the grammar refinement process is provided. The first column contains the number of refinement iterations passed. In the table cells, there are numbers of files from each project which still cause parsing failure. Having started with the smallest project, the LanD itself, we included the bigger ones to the testing process as the grammar became refined enough to produce parser capable to parse all the files under consideration. For two refinement iterations, the number of errors for LanD and PascalABC.NET was reduced to zero. Surprisingly, even so we got a significant number of erroneously parsed and unparsed files for Roslyn (209 files out of 8027). Analyzing them we found out that it was caused by tuple types and tuple literals. It is one of the new features added to C# 7.0. These constructs may look exactly like method arguments, causing confusion during parsing. The problem was solved by the less restrictive class member patterns description: the entire header is matched by the `name` pattern which includes the `arguments` pattern. The `arguments` pattern matches method arguments as well as tuple types. A more accurate division of `name` into modifiers, a type, an entity name and arguments was moved at the automatically built syntax tree post-processing stage. Expression bodied properties became another cause of errors. They are widely used in Roslyn but are not presented in LanD and PascalABC.NET. To process them as the water, the `init_expression` anti-pattern was added and the `init_value` anti-pattern was refined.

At the last iteration of grammar testing and refinement, the number of errors is still non-zero. However, on closer inspection it was proved to be not a consequence of inaccuracies in the grammar structure. The first file[6] is a test file for the Roslyn compiler, it contains the text of the program in Shift-JIS encoding, which is used for Japanese, moreover, the class name is written in Japanese. The latter causes a lexical

[6] https://github.com/dotnet/roslyn/blob/master/src/Compilers/Test/Resources/Core/Encoding/sjis.cs

analysis error. We consider the usage of national alphabets for entity naming to be a rare case, but, if necessary, the `ID` token can be adopted as needed. The second file[7] also belongs to the testing infrastructure, it contains a meta-information in a form of invalid global code: there is a string field, declared directly inside the namespace but outside of the class. In the third file[8], the code containing `using` directives and a class definition is placed after the namespace definition. This code is enclosed in `#if false` preprocessor directive, so it is not compiled after the preprocessing stage. Our tolerant parser works with the pure sources and ignores the directives, so it justifiably treats this program as incorrect.

The resulting C# grammar is aimed at all-encompassing parsing of all the possible valid C# code variations from three real-world software projects, at the same time it is both tolerant with respect to code in places indicated with `Any`, and lightweight. For instance, the baseline C# parser description[9] for the industrial compiler generator ANTLR, which uses an extended LL(*) algorithm [22], contains 1159 lines, and lexical analyzer specification contains 1101 lines. The text of our tolerant LL(1) C# grammar has (including token definitions and different generator options) just 51 lines. Developing a parser for a certain project, one can make the grammar even more lightweight if some project-specific restrictions are known. In case some coding conventions are applied, land and water content become less variable. If a legacy code is parsed, one can be sure that the latest language features are not in use there, so the grammar is allowed not to contain patterns and anti-patterns for them.

At the next stage of the experiment, the syntax trees of the parsed files were used to calculate the numbers of successfully discovered LanD entities that we are interested in, solving the code markup task. As control numbers, the results of counting the same entities using syntax trees built by Roslyn were used. The entities were grouped into five categories: enums, classes, fields, properties, methods. The grouping is carried out in accordance with the hierarchy of classes representing the nodes of a syntactic tree in Roslyn. Entities which corresponds to Roslyn tree nodes of type `BaseFieldDeclarationSyntax` are marked as fields. These are fields themselves, as well as events described without access methods. Elements corresponding to nodes of types inherited from `BasePropertyDeclarationSyntax` are treated as properties. In addition to properties themselves, these are indexers and events with explicitly specified `add` and `remove` accessors. Methods correspond to `BaseMethodDeclarationSyntax` type: it is the parent type for method, constructor, destructor, and operator nodes.

---

[7] https://github.com/dotnet/roslyn/blob/master/src/Compilers/Test/Resources/Core/SymbolsTests/Metadata/public-and-private.cs

[8] https://github.com/dotnet/roslyn/blob/master/src/Workspaces/Core/Portable/Shared/Extensions/ObjectExtensions.cs

[9] https://github.com/antlr/grammars-v4/tree/master/csharp

In Table 4, the quantitative results are presented. For all projects in all categories, LanD detects more entities than Roslyn. The difference is caused by the conditional compilation directive `#if`, which is actively used in the projects under consideration. For example, in PascalABC.NET the `#if DEBUG` construct is widely used to enable debug output and additional information collecting, conditional compilation is also presented in the sources of the syntax analyzers, which are generated with GPPG.

Table 4. Number of entities found by Roslyn/LanD.

	Enums	**Classes**	**Fields**	**Properties**	**Methods**
LanD	13/14	94/95	390/390	248/253	431/436
PABC	356/363	4611/4622	16720/16753	12326/12350	42248/42386
Roslyn	437/441	21583/21622	19606/19737	21886/21919	108040/108400

Roslyn parser has an integrated preprocessor which resolves `#if` conditions and pass to the parsing stage only the appropriate parts of the code. LanD is a language-independent tool, so it does not have a built-in notion of directives. For a lightweight parser, directives are defined as single-line lexemes which are usually skipped. As a result, LanD statistics take into account all the entities regardless of whether or not they are enclosed in the `#if` directive with an undefined symbol. It should be noted that C# preprocessing is a fairly simple task. If necessary, the correct preprocessor can be easily written and applied to the text passed to the LanD-generated C# parser. However, this will lead to a loss of information about the areas excluded by the preprocessor.

7. Conclusion

In the present paper, the LL(1) parsing algorithm modification is proposed. This modification is intended for performing tolerant parsing based on the island grammars technique. The special `Any` symbol is integrated into the algorithm to add a capability to match token sequences which are not explicitly described in the grammar. With regard to island grammar development, the presence of `Any` simplifies the description of water and partially eliminates the need to describe the structure and variations of irrelevant areas. Besides, `Any` can be used for relevant code description in case this code contains lakes — areas for which we are interested only in pure token sequence, not in the structural information. Our `Any` implementation fixes the shortcomings of the closest analogues. It is more accurate and less restrictive in comparison with Coco/R and LightParse parser generators, it is also more simple than bounded seas approach, and still powerful enough to parse sources of large-scale software projects. It is experimentally proved that the lightweight parser of the C# language with built-in automatic construction of the syntax tree, which was developed by the authors of the current paper, makes it possible to successfully analyze the source codes of industrial software products and provides one hundred percent finding of points of interest. The developed generator of lightweight parsers is planned to be used in solving the sustainable code markup problems.

Tolerant grammar description and syntax tree post-processing are supposed to be simplified by integrating the *Schrödinger's token* concept [13] into lexical and syntax analyzers. In particular, it can be useful for analyzing C# language where, along with reserved keywords, there are contextual keywords. Some of them (for example, words `where` and `partial`) directly affect the separation of land and water and the land structure analysis. Possible directions for further research are also a more intelligent resolution of the consecutive `Any` problem and integration of the `Any` symbol into LR(1) parsing algorithm.

References

[1]. Goloveshkin A.V. Searching and analysing crosscutting concerns in marked up programming language grammar. Izvestija vuzov. Severo-Kavkazskij region. Tehnicheskie nauki [University News. North-Caucasian Region. Technical Sciences Series], 2017, issue 3, pp. 29–34 (in Russian). DOI: 10.17213/0321-2653-2017-3-29-34.

[2]. Afroozeh A., Bach J.-C., van den Brand M., Johnstone A., Manders M., Moreau P.-E., Scott E. Island grammar-based parsing using GLL and Tom. Software Language Engineering: 5th International Conference, SLE 2012, Dresden, Germany, September 26-28, 2012, Revised Selected Papers. Springer Berlin Heidelberg, 2013, pp. 224–243.

[3]. Van den Brand M., Sellink M.P.A., Verhoef C. Obtaining a COBOL grammar from legacy code for reengineering purposes. In Proceedings of the 2nd International Conference on Theory and Practice of Algebraic Specifications. BCS Learning & Development Ltd., 1997, pp. 6–16.

[4]. Moonen L. Generating robust parsers using island grammars. In Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01). IEEE Computer Society, 2001, pp. 13–22.

[5]. Moonen L. Lightweight impact analysis using island grammars. In Proceedings of the 10th International Workshop on Program Comprehension (IWPC). IEEE Computer Society, 2002, pp. 219–228.

[6]. Graham S.L., Haley C.B., Joy W.N. Practical LR error recovery. SIGPLAN Notes, vol. 14, issue 8, 1979, pp. 168–175.

[7]. Burke M.G., Fisher G.A. A practical method for LR and LL syntactic error diagnosis and recovery. ACM Trans. Program. Lang. Syst., vol. 9, issue 2, 1987, pp. 164–197.

[8]. De Jonge M., Nilsson-Nyman E., Kats L.C.L., Visser E. Natural and flexible error recovery for generated parsers. Software Language Engineering: Second International Conference, SLE 2009, Denver, CO, USA, October 5-6, 2009, Revised Selected Papers. Springer Berlin Heidelberg, 2010, pp. 204–223.

[9]. Nilsson-Nyman E., Ekman T., Hedin G. Practical scope recovery using bridge parsing. Software Language Engineering: First International Conference, SLE 2008, Toulouse, France, September 29-30, 2008. Revised Selected Papers. Springer Berlin Heidelberg, 2009, pp. 95–113.

[10]. Koppler R. A systematic approach to fuzzy parsing. Software: Practice and Experience, vol. 27, issue 6, 1997, pp. 637–649.

[11]. Carvalho P., Oliveira N., Henriques P.R. Unfuzzying fuzzy parsing. 3rd Symposium on Languages, Applications and Technologies, ser. OpenAccess Series in Informatics (OASIcs), vol. 38, 2014, pp. 101–108

[12]. S. Klusener and R. Lämmel, Deriving tolerant grammars from a base-line grammar. In Proceedings of the International Conference on Software Maintenance. IEEE Computer Society, 2003, pp. 179–188.

[13]. Aycock J., Horspool R.N., Schrödinger's token. Software: Practice and Experience, vol. 31, issue 8, 2001, pp. 803–814.

[14]. Grune D., Jacobs C.J. Parsing Techniques: A Practical Guide (2nd Edition). Springer-Verlag, New York, 2008, 662 p.

[15]. Scott E., Johnstone A. GLL parsing. Electron. Notes Theor. Comput. Sci., vol. 253, issue 7, 2010, pp. 177–189.

[16]. Mössenböck H. (2014) The compiler generator Coco/R. Available at: http://ssw.jku.at/Coco/Doc/UserManual.pdf, accessed 02.03.2018.

[17]. Malevannyy M. Lightweight parsing and its application in development environment. Informatizatsiya i svyaz [Informatization and communication], 2015, vol. 3, pp. 89–94 (in Russian).

[18]. Malevannyy M.S., Mikhalkovich S.S. Context-based model for concern markup of a source code. Trudy ISP RAN/Proc. ISP RAS, 2016, vol. 28, issue 2, pp. 63–78. DOI: 10.15514/ISPRAS-2016-28(2)-4.

[19]. Kurš J., Lungu M., Iyadurai R., Nierstrasz O. Bounded seas. Comput. Lang. Syst. Struct., 2015, vol. 44, pp. 114–140

[20]. Ford B. Packrat parsing: Simple, powerful, lazy, linear time. Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming, ser. ICFP '02. ACM, 2002, pp. 36–47.

[21]. Aho A.V., Lam M.S., Sethi R., Ullman J.D. Compilers: Principles, Techniques, and Tools (2nd Edition). Addison-Wesley Longman Publishing Co., Inc., 2006, 1000 p.

[22]. Parr T., Harwell S., Fisher K. Adaptive LL(*) parsing: The power of dynamic analysis. SIGPLAN Notes, vol. 49, issue 10, 2014, pp. 579–598.

Толерантный синтаксический анализ с использованием специального символа «Any»: алгоритм и практическое применение

А.В. Головешкин <alexeyvale@gmail.com>
С.С. Михалкович <miks@sfedu.ru>
Институт математики, механики и компьютерных наук им. И.И. Воровича,
Южный федеральный университет,
344090, Россия, г. Ростов-на-Дону, ул. Мильчакова, д. 8а

Аннотация. Толерантный синтаксический анализ позволяет найти области программы, представляющие интерес в контексте конкретной задачи, и извлечь информацию об их структуре. В то время как эти области должны быть подробно описаны в грамматике языка, другие части программы могут быть не описаны совсем или описаны менее детально, при этом генерируемый парсер должен признавать корректными все возможные вариации программы в нерелевантных областях, то есть, должен быть толерантным по отношению к ним. Островные грамматики — один из основных способов реализации толерантного парсинга. Термином «остров» обозначаются релевантные области кода, термином «вода» — нерелевантный код. В настоящей работе описывается модифицированный LL(1) алгоритм со встроенной обработкой специального символа «Any», позволяющего сопоставлять последовательности токенов, не описанные разработчиком грамматики в явном виде. Применение данного алгоритма к островным грамматикам ведёт к сокращению описания воды и упрощению описания островов. Наша реализация «Any» является более безопасной для использования и менее ограничительной по сравнению с ближайшими аналогами в генераторах Coco/R и LightParse. Также она более предсказуема и требует меньших накладных расходов в сравнении с концепцией «ограниченных морей», внедрённой в PetitParser. На базе алгоритма реализован генератор компиляторов со встроенным языком описания островных грамматик. Как показано в разделе экспериментов, сгенерированный по островной грамматике языка C# толерантный парсер может быть успешно применён для анализа крупных промышленных проектов.

Ключевые слова: толерантный парсинг; устойчивый парсинг; легковесный парсинг; частичный парсинг; островная грамматика; генерация парсеров

DOI: 10.15514/ISPRAS-2018-30(4)-1

Для цитирования: Головешкин А.В., Михалкович С.С. Толерантный синтаксический анализ с использованием специального символа «Any»: алгоритм и практическое применение. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 7-28 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(4)-1

Список литературы

[1]. Головешкин А.В. Поиск и анализ сквозных функциональностей в размеченной грамматике языка программирования. Известия вузов. Северо-Кавказский регион. Технические науки, 2017, вып. 3, стр. 29–34. DOI: 10.17213/0321-2653-2017-3-29-34.

[2]. Afroozeh A., Bach J.-C., van den Brand M., Johnstone A., Manders M., Moreau P.-E., Scott E. Island grammar-based parsing using GLL and Tom. Software Language Engineering: 5th International Conference, SLE 2012, Dresden, Germany, September 26-28, 2012, Revised Selected Papers. Springer Berlin Heidelberg, 2013, pp. 224–243.

[3]. Van den Brand M., Sellink M.P.A., Verhoef C. Obtaining a COBOL grammar from legacy code for reengineering purposes. In Proceedings of the 2nd International Conference on Theory and Practice of Algebraic Specifications. BCS Learning & Development Ltd., 1997, pp. 6–16.

[4]. Moonen L. Generating robust parsers using island grammars. In Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE’01). IEEE Computer Society, 2001, pp. 13–22.

[5]. Moonen L. Lightweight impact analysis using island grammars. In Proceedings of the 10th International Workshop on Program Comprehension (IWPC). IEEE Computer Society, 2002, pp. 219–228.

[6]. Graham S.L., Haley C.B., Joy W.N. Practical LR error recovery. SIGPLAN Notes, vol. 14, issue 8, 1979, pp. 168–175.

[7]. Burke M.G., Fisher G.A. A practical method for LR and LL syntactic error diagnosis and recovery. ACM Trans. Program. Lang. Syst., vol. 9, issue 2, 1987, pp. 164–197.

[8]. De Jonge M., Nilsson-Nyman E., Kats L.C.L., Visser E. Natural and flexible error recovery for generated parsers. Software Language Engineering: Second International Conference, SLE 2009, Denver, CO, USA, October 5-6, 2009, Revised Selected Papers. Springer Berlin Heidelberg, 2010, pp. 204–223.
[9]. Nilsson-Nyman E., Ekman T., Hedin G. Practical scope recovery using bridge parsing. Software Language Engineering: First International Conference, SLE 2008, Toulouse, France, September 29-30, 2008. Revised Selected Papers. Springer Berlin Heidelberg, 2009, pp. 95–113.
[10]. Koppler R. A systematic approach to fuzzy parsing. Software: Practice and Experience, vol. 27, issue 6, 1997, pp. 637–649.
[11]. Carvalho P., Oliveira N., Henriques P.R. Unfuzzying fuzzy parsing. 3rd Symposium on Languages, Applications and Technologies, ser. OpenAccess Series in Informatics (OASIcs), vol. 38, 2014, pp. 101–108
[12]. S. Klusener and R. Lämmel, Deriving tolerant grammars from a base-line grammar. In Proceedings of the International Conference on Software Maintenance. IEEE Computer Society, 2003, pp. 179–188.
[13]. Aycock J., Horspool R.N., Schrödinger's token. Software: Practice and Experience, vol. 31, issue 8, 2001, pp. 803–814.
[14]. Grune D., Jacobs C.J. Parsing Techniques: A Practical Guide (2nd Edition). Springer-Verlag, New York, 2008, 662 p.
[15]. Scott E., Johnstone A. GLL parsing. Electron. Notes Theor. Comput. Sci., vol. 253, issue 7, 2010, pp. 177–189.
[16]. Mössenböck H. (2014) The compiler generator Coco/R. Available at: http://ssw.jku.at/Coco/Doc/UserManual.pdf, accessed 02.03.2018.
[17]. Малёванный М.С. Легковесный парсинг и его использование для функций среды разработки. Информатизация и связь, 2015, том 3, стр. 89–94.
[18]. Malevannyy M.S., Mikhalkovich S.S. Context-based model for concern markup of a source code. Trudy ISP RAN/Proc. ISP RAS, 2016, vol. 28, issue 2, pp. 63–78. DOI: 10.15514/ISPRAS-2016-28(2)-4.
[19]. Kurš J., Lungu M., Iyadurai R., Nierstrasz O. Bounded seas. Comput. Lang. Syst. Struct., 2015, vol. 44, pp. 114–140
[20]. Ford B. Packrat parsing: Simple, powerful, lazy, linear time. Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming, ser. ICFP '02. ACM, 2002, pp. 36–47.
[21]. Aho A.V., Lam M.S., Sethi R., Ullman J.D. Compilers: Principles, Techniques, and Tools (2nd Edition). Addison-Wesley Longman Publishing Co., Inc., 2006, 1000 p.
[22]. Parr T., Harwell S., Fisher K. Adaptive LL(*) parsing: The power of dynamic analysis. SIGPLAN Notes, vol. 49, issue 10, 2014, pp. 579–598.

An Interactive Specializer Based on Partial Evaluation for a Java Subset

[1] *I. A. Adamovich <i.a.adamovich@gmail.com>*
[2] *And. V. Klimov <klimov@keldysh.ru>*
[1] *Ailamazyan Program Systems Institute of Russian Academy of Sciences, 4a Peter the First str., Veskovo, Yaroslavl region, 152021, Russia*
[2] *Keldysh Institute of Applied Mathematics of Russian Academy of Sciences, 4 Miusskaya sq., Moscow, 125047, Russia*

Abstract. Specialization is a program optimization approach that implies the use of a priori information about values of some variables. Specialization methods are being developed since 1970s (mixed computations, partial evaluation, supercompilation). However, it is surprising, that even after three decades, these promising methods have not been put into the wide programming practice. One may wonder: What is the reason? Our hypothesis is that the task of specialization requires much greater human involvement into the specialization process, the analysis of its results and conducting computer experiments than in the case of common program optimization in compilers. Hence, specializers should be embedded into integrated development environments (IDE) familiar to programmers and appropriate interactive tools should be developed. In this paper we provide a work-in-progress report on results of development of an interactive specializer based on partial evaluation for a subset of the Java programming language. The specializer has been implemented within the popular Eclipse IDE. Scenarios of the human-machine dialogue with the specializer and interactive tools to compose the specialization task and to control the process of specialization are under development. An example of application of the current version of the specializer is shown. The residual program runs several times faster than the source one.

Keywords: program analysis, program transformation, interactive program specialization, partial evaluation, object-oriented language, integrated development environment.

DOI: 10.15514/ISPRAS-2018-30(4)-2

For citation: Adamovich I.A., Klimov And.V. An Interactive Specializer Based on Partial Evaluation for a Java Subset. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 4, 2018. pp. 29-44. DOI: 10.15514/ISPRAS-2018-30(4)-2

---

[1] Supported by RFBR research project No. 18-37-00454 (contribution: development of interactive methods of partial evaluation, design of the architecture and implementation of the specializer, analysis of related works).
[2] Supported by RFBR research project No. 16-01-00813 (contribution: problem statement, design of methods based on the existing approaches, supervision, analysis of related works).

1. Introduction

The method of program specialization known as *partial evaluation* was invented more than 30 years ago along with the achievement of the famous result [1], [2] of evaluation of the First, Second and Third Futamura projections [3]–[5] for a tiny List subset. The first round of research was completed in early 1990s when the main textbook on partial evaluation had been published [2]. A lot of programming problems were found to be solved by program specialization (the most known being the generation of a compiler from an interpreter by the Second Futamura Projection) and the emergence of a new class of program development tools based on specialization were expected. Some other program specialization techniques, e.g., *supercompilation* [6], [7], has been developed in parallel as well. However, it is surprising that even after three decades these promising methods have not been put into the wide programming practice. One may wonder: What is the reason?

Our hypothesis is that the main expectation that governed the development of specializers was wrong. The developers of these methods hoped that specializers could work in fully automatic mode and they just needed to invent some finitely many features and improvements that solve the problem, after which “the great goal” would be achieved and happy programmers started using the new tools. They expected that specializers could work in the similar “black-box mode” as optimizing compilers. However this did not happen. The time and space complexity of the program transformations that were necessary for specialization, turned out to be much higher than the complexity of program optimizations that can be used as “black boxes” with short and predictable run time and consumed memory.

We argue that automatic methods of program optimization have reached certain inherent limits. In order to develop and use more powerful tools, we must give up the expectations that the program analysis and transformation systems will operate in automatic mode without human intervention. Program specializers possess too many degrees of freedom and choice, which cannot be resolved by the algorithms of their kind and, therefore, should use human help.

Based on this observation, we put forward the goal of construction of an interactive specializer embedded in a habitual integrated development environment (IDE) such as Eclipse [8]. Eclipse provides a rich open-source toolkit referred to as Java development tools (JDT) [9], which allows a developer to deal only with essential tasks of analysis, visualization and transformation of Java code. Adequate human-machine dialogue tools to control the specializer and deal with the results of specialization are to be developed. We would like to emphasize that there is strict separation of concerns between the machine and the human: the specializer guaranties the functional equivalence of program transformation and the user is responsible for the control of the specializer in such a way that it produces the code that satisfied user’s goals and needs (which the machine does not know).

```
public class AckermannExample {
    public final static long A (long x, long y) {
        if (x == 0) return y + 1;
        else if (y == 0) return A(x - 1, 1);
        else return A(x - 1, A(x, y - 1));
    }

    @Specialize
    public static long test(long y) {
        return A(3, y);
    }
}
```

Fig.1. Source code of Ackermann function

We think that partial evaluation is better suited than other specialization methods (like supercompilation) for human-machine dialogue organized in such a way that the user comprehends what is happing in the specializer, receives valuable and interesting information about his code, is capable of adjusting the source code to be better specialized and controls the specializer. The reason is that the method of partial evaluation consists of two stages:

- *binding-time analysis* (BTA) of source code that selects the parts of the code that are to be evaluated at specialization time, and
- *residual program generation* (RPG) that follows the information supplied by BTA, performs specialization proper and produces the resulting code (referred to as *residual*).

A pleasant feature of BTA is that its result (called *BT annotation*) may be naturally shown on the source code by highlighting and due to such visualization the residual code is intuitively predictable. We hope that this will allow for easy adoption of specializers as new programming tools by rank-and-file programmers.

Terminological remark. In the theory of partial evaluation the parts of source code to be evaluated during specialization are called *static*. The other source code that is transferred to the residual program (*residualized*) is referred to as *dynamic*. The term *static* conflicts with the `static` modifier in Java and the term *dynamic* may be confused with the run-time notions. That is why we avoid using these words in the partial evaluation sense and use abbreviations `S` and `D` instead, e.g., `S`-annotation, `D`-annotation, `S`-code, `D`-code, `S`-part and `D`-part of a program.

The contributions of this paper are as follows.

- We show the first results of development of the Java specializer, where partially evaluated code is restricted to operations on primitive types.
- We demonstrate the work of the specializer by an example of specialization of the Ackermann function with respect to the first argument.
- We discuss some of the details of implementation in Eclipse and the methods and features to be implemented in future.

```
public class AckermannExample {
    public long test(long y) {
        return A_3(y);
    }

    public final long A_3(long y) {
        if (y == 0) return A_2(1);
        else return A_2(A_3(y - 1));
    }

    public final long A_2(long y) {
        if (y == 0) return A_1(1);
        else return A_1(A_2(y - 1));
    }

    public final long A_1(long y) {
        if (y == 0) return A_0(1);
        else return A_0(A_1(y - 1));
    }

    public final long A_0(long y) {
        return y + 1;
    }
}
```

Fig. 2. Residual code of Ackermann function

The outline of the paper is as follows. In Section 2 we present the basics of partial evaluation for Java by an example of specialization of the Ackermann function. In Section 3 a bird-eye view of the implementation of the specializer in the Eclipse IDE is presented. Section 4 contains a survey of related works in comparison with our specializer. In Section 5 we conclude.

2. Java Specialization by Example

Fig. 1 and 2 contain screenshots of the source and residual code of the Ackermann function made from the running specializer in Eclipse IDE.

The method `A` implements the Ackermann function and the method `test` invokes it with the first constant argument `3`. The Java annotation `@Specialize` at the method `test` specifies that it should be specialized, i.e., its body is to be replaced with the residual code and the specialized versions of the methods that it invokes are to be generated and added to the program. The names of the methods `A` and `test` in their headers are marked in orange in order to show that they are involved in BTA. The bodies of these methods are analyzed and annotated: green highlighting marks `S`-parts of code. (You see gray highlighting in fig. 1 if you read this paper in a monochrome print).

2.1. Binding-Time Analysis

The BTA algorithm for variables and operations of primitive types is rather straightforward. First, all constants are annotated with `S`. Then recursively: a subexpression containing only `S`-parts becomes `S`; a local variable declaration and an assignment with `S` right-hand sides become `S`; a method parameter that correspond to `S` arguments at all points of invocation becomes `S`; in case of conflict of several invocations of the same method the conflicting parameter becomes `D`; a conflict on several assignments to a local variable turns it to `D` as well; an `if` statement with the `S` conditional expression is annotated with `S` regardless of the annotation of its branches (this means that `if-else` will disappear while one of the branches will be residualized); other control statements are analyzed and annotated similarly. When the recursion reaches the fixed point, the remaining parts of code are annotated with `D`. `D`-parts are not highlighted in Figure 1.

This mode of operation of BTA, when each code fragment gets univocal annotation `S` or `D`, is referred to as *monovariant*. The more general mode when several versions of annotation are allowed is called *polyvariant*. The current version of BTA is monovariant. In future we plan to implement polyvariant BTA for classes and reference types according the theory developed in [10]–[18].

Monovariant BTA on primitive types can be defined formally as abstract interpretation on a lattice with 3 elements: *undefined* < `S` < `D`.

As an illustration of monovariance, notice that in figure 1 method `A` is invoked 3 times in the source code, one of which has both `S` arguments, another 2 invocations have the first `S` argument and the second one is `D`. The first invocation is processed in the same way as the other two with the second `S` argument assigned to the `D` formal parameter.

2.2. Residual Program Generation

At the generation stage, partial evaluation starts from the method with the `@Specialize` annotation and recursively visits all invoked methods in turn. Notice that, since all statements and methods with side effects are considered `D` and hence are residualized rather than executed at specialization time, the order of specialization of methods does not matter. For each of the specialized methods, several residual versions can be produced — one for each combination of values of `S` arguments. They got different names of the form (in the current version): *source-name_number*. They have only those parameters that correspond to `D` parameters in the source code.

The current version of the specializer can loop forever if infinitely many values of `S` arguments are generated. The production version of the specializer should contain special debugging means to gracefully leave such situations. This is our future work. In Figure 2 there are 4 versions of residual method `A` corresponding to values `0`, `1`, `2`, `3` of its first argument. Notice that because of monovariance the invocations

`A_2(1)`, `A_1(1)`, and `A_0(1)` have not being evaluated, since the constant `1` correspond to the `D` parameter of method `A`.

2.3. Running Source and Residual Programs

We have chosen this example for presentation, since it demonstrates all main features of the current version of the specializer. We did not expect a significant speed-up as it seemed that asymptotically the number of method invocations was almost the same and the invocations were the most expensive operations in this example. Thus we were very surprised when the speed-up was about 3 times.

The obtained acceleration can be explained by several reasons. First, calculation showed that the specialized version performs 1.86 times less Java byte code instructions. Second and more important, it is natural to suppose that the JIT compiler in JVM performs inlining of those specialized method that are simpler and more compact than in the source code.

This example illustrates the principle, which we observed many times in experiments with various specializers: a specializer does not replace the classic optimizing compilers. Rather, we observe “composition” of optimizations by a specializer and a low-level optimizing compiler and hence multiplication of speed-ups. Residual code produced by specializers is more amendable for classic optimizations than code written by a human being. We may conclude that specialization opens up additional opportunities for program optimization.

3. Architecture of Specializer

The specializer has been implemented in the Eclipse development environment (IDE) [8]. The IDE has open source code and provides points and tools to extend it.

The basis for Eclipse extension is the concept of a plug-in. Each plug-in is an archive JAR file containing a so-called manifest, a set of files describing the dependencies of the plug-in and the possibility of its extension (extension points). Other plug-ins can add their functionality to these extension points. For example, one might want to add his toolbar extensions to an already implemented toolbar plug-in.

A small tool is usually implemented as a one plug-in, while a large one is often provided as a set of plug-ins. Our specializer is implemented as three Eclipse plug-ins.

The specializer consists of the following plug-ins:

- a plug-in SpecCore is the core of the specializer, which implements its main functionality;
- a plug-in SpecMarkers is responsible for text highlighting in the Eclipse editor in accordance with the annotation produced by the SpecCore plug-in;
- a plug-in SpecMenus implements interactions with various menus (including context menus) and toolbars to provide a user-friendly interface.

The SpecCore implements the binding-time analysis (BTA) and the generation of a residual program. When analyzing the source program the plug-in SpecCore uses the abstract syntax tree (AST) built by the Eclipse Java development tools (JDT).

JDT is a set of plug-ins that provides us with an easy way to manipulate Java source code.

The second of the three plug-ins that form the specializer is the SpecMarkers plug-in. It is responsible for highlighting the source code, which allows the programmer to see which parts of the program are evaluated at specialization time and which are residualized. This helps him to understand how to change the code to provide better specialization.

The last part of the specializer is the SpecMenus plug-in. This plug-in uses the extension points of other plug-ins to add the necessary elements to some menus. It adds two new buttons to the main toolbar of Eclipse: Enable/Disable the highlighting and the "Generate optimized Java files" button. Also this plug-in adds items to the context menu of the Project Explorer and Package Explorer views.

4. Related Work and Comparison

A lot of works are devoted to partial evaluation for functional languages. The book [2] summarizes the first wave of development of this method.

Later on, research into partial evaluation for imperative "Algol-like" languages [19], [20] and C [21] was performed. In early 1990's, the first (to our knowledge) specializer for C was developed, called C-MIX [21], [22]. Chapter 11 of the book [2] contains its detailed presentation. C-MIX specializes a program in three stages.

The first stage is the analysis of references. For each reference variable, a set of the variables that it could refer to is built. If the analysis finds that several reference variables can refer to the same memory, they are labeled identically. The second stage is the construction of a binding-time annotation of the source code. References to the same memory area are annotated identically. In case of conflicts, the annotation is reduced to `D` as usual. The third stage is the generation of the residual program.

Specialization of reference types in Java can be similar to elaboration of pointers in C-MIX. However, Java stricter typing and managed run-time can be leveraged for deeper specialization. The current version of our specializer annotates all reference variables `D` and, therefore, they are left unchanged. Our future work is to add the binding-time analysis of reference types. Unlike C-MIX, we expect that our specializer will still work in two stages — without the reference analysis phase.

Further development of ideas of C-MIX led to the creation of a specializer of programs written in C, called Tempo [23], [24]. This specializer is much like C-MIX.

The next important step was the development of the first specializer for an object-oriented language — JSpec for Java [25]. JSpec uses the Harissa compiler [26] to translate the Java program into C. Then the Tempo specializer mentioned above transforms the program. The obtained C-representation of a specialized Java program is mapped back into Java using the Assirah translator [25]. Finally, the AspectJ tool weaves the specialized program with the source program to get the executable Java bytecode. The main limitation of JSpec is that it is capable of partially evaluating only immutable classes and objects, while mutable objects are always residualized. Our goal is to waive this restriction.

The most advanced (to our knowledge) partial evaluation method for object-oriented languages like C# and Java has been developed in CILPE [10]–[18], a partial evaluator for Common Intermediate Language (CIL), the bytecode of the Microsoft .NET Framework. It supports almost all of the basic constructs of object-oriented languages such as C# and Java. In CILPE, a new concept of a binding-time heap (BT heap) has been introduced. A BT heap is an abstract description of the state of a run-time heap, which allows us to separate reference type data into evaluated at specialization time and residualized ones and to avoid the use of the reference analysis stage as in C-MIX. As a result of specialization, some of the objects are no longer created in the residual program, and if necessary, local variables are used instead of object fields. We will base on the results of this research in our future work to implement BTA of classes and partial evaluation of objects.

A relatively new specializer of Java programs is Civet [27]. Civet is based on a so-called Hybrid Partial Evaluation (HPE) approach. Specialization in HPE is performed in *online* mode, i.e., in one pass, while the programmer can specify which parts of the program have `S`-annotation. On the contrary, in our specializer we choose the *offline* approach, i.e., the residual program is built at the stage of generation of the residual program after the completion of the binding-time analysis, where information about the `S`-parts of the program is collected automatically rather than specified by the user as in Civet[1].

PE-KeY [28] is a partial evaluator for Java programs based on the KeY verification system [29]. PE-Key works in two stages. At the first stage, the program is executed in a symbolic form with the application of a special set of rules. At the second stage, a residual program is synthesized, while the rules are applied in the opposite direction. The PE-KeY approach is similar to the classical offline specialization that our specializer uses: a specialized program is produced in two stages. However, in the first stage of PE-KeY, the program is executed symbolically, while our binding-time analysis performs an abstract interpretation of the program. In addition, due to limitations of the KeY verification system, PE-KeY does not support floating-point arithmetic, while our specializer supports.

JSpec, Civet, PE-Key deal with objects at specialization time, while the current version of our specializer annotates classes and variables of reference types with `D`

[1] For discussion of the features of and differences between online and offline partial evaluation see [2, Chapter 7].

and thus residualizes them unchanged. The extension of our specializer to partial evaluation of classes and objects is our future work.

The specializers considered above interact with the user through the command line, so it's extremely difficult to use them. In order for the specialization to be widely used, it is required to develop the methods of interaction with the user and to embed the specializer into an integrated development environment convenient for the programmer, what we are implementing in our specializer. This is a crucial difference.

We know about just one work on partial evaluation carried out in a practical setting – the GraalVM toolkit in Oracle Labs [30], [31]. The toolkit is designed for defining domain-specific languages by interpreters and, nevertheless, achieving high-performance by using a specializer. The first Futamura projection provides an opportunity for such acceleration (see [3], [4] and [2, Chapter 1.5.1]): given a program and an interpreter that executes the program, GraalVM specializes the interpreter with respect to a part of the given program and produces the machine code of this part. The resulting code is executed much faster than the original one in the interpreter. The main goal of GraalVM is to provide a technology similar to just-in-time (JIT) compilation for the developer of a programming language without the need to implement the complex machinery of JIT. The interpreter specialization in GraalVM is not automatic and uses prompts by the interpreter developer. This case of implementation of partial evaluation confirms that practical application of specialization requires guidance from the programmer. We conduct our research in the same direction: methods and tools are being developed to provide the programmer with information about program behavior under specialization and levers to control the partial evaluation processes.

5. Conclusion

In this paper we put forward the task of development of an interactive specializer.

We argue that the current stage of program specialization methods has reached certain limits because the previously implemented specializers do not imply the participation of the user in the process of specialization. Our specializer uses the offline partial evaluation approach, where the program transformation if performed in two stages — binding-time analysis (BTA) and residual program generation (RPG). We briefly described the architecture of our interactive specializer in the Eclipse development environment.

We illustrated the work of the specializer with the famous example of the Ackermann function and the result of its specialization with respect to its first argument. The specialized program runs several times (about three) faster than the original one.

We see the following directions for further development of the specializer:

- to develop and implement binding-time analysis and residual program generation for classes and objects;

- to implement interactive tools for composing a specialization task and controlling the process of binding-time analysis and residual program generation;
- to implement tools to visualize the correspondence between source and residual code;
- to demonstrate that a well-developed specializer can convert well-structured high-level human-oriented code, which can not be automatically parallelized, into code that can be parallelized by existing methods and tools;
- to prepare demo programs that benefit from specialization, for example, building a compiler from an interpreter;
- to generalize the binding-time analysis from monovariant to polyvariant;
- to develop an interactive tracer (similar to run-time debuggers) that allows the user to observe the analysis and generation processes in order to improve the behavior of his code under specialization.

Availability. The current version of our specializer is available at ftp://ftp.botik.ru/rented/iaadamovich/Specializer/.

Acknowledgment

We are grateful to our friends and colleagues Yuri Klimov, Arkady Klimov, Sergei Romanenko, Sergei Abramov for valuable advices on specialization methods in general and partial evaluation in particular and constructive feedback on the design of our specializer system.

References

[1]. Jones N.D., Sestoft P. and Søndergaard H. An experiment in partial evaluation: the generation of a compiler generator. Rewriting Techniques and Applications, Lecture Notes in Computer Science, J.-P. Jouannaud, (Ed.), vol. 202. Springer-Verlag, 1985, pp. 124–140

[2]. Jones N.D., Gomard C.K., and Sestoft P. Partial Evaluation and Automatic Program Generation. Prentice-Hall, 1993, 415 p. Available at: http://www.itu.dk/~sestoft/pebook/pebook.html, accessed 20.06.2018

[3]. Futamura Y. Partial evaluation of computation process — an approach to a compiler-compiler. Systems, Computers, Controls, vol. 2, no. 5, 1971, pp. 45–50

[4]. Futamura Y. Partial evaluation of computation process — an approach to a compiler-compiler. Higher-Order and Symbolic Computation, vol. 12, no. 4, Dec 1999, pp. 381–391. Updated and revised version of [3]. Available at: http://doi.org/10.1023/A:1010095604496, accessed 20.06.2018

[5]. Futamura Y. EL1 Partial Evaluator (Progress Report). Center for Research in Computing Technology, Harvard University, Tech. Rep., 1973. Available at: http://fi.ftmr.info/PE-Museum/EL1.PDF, accessed 20.06.2018

[6]. Turchin V.F. The concept of a supercompiler. ACM Transactions on Programming Languages and Systems, vol. 8, no. 3, 1986, pp. 292–325

[7]. Turchin V.F. Supercompilation: techniques and results. Perspectives of System Informatics, Second International Andrei Ershov Memorial Conference, Akademgorodok, Novosibirsk, Russia, June 25-28, 1996. Proceedings, Lecture Notes in Computer Science, D. Bjørner, M. Broy, and I.V. Pottosin, (Eds.), vol. 1181. Springer, 1996, pp. 227–248

[8]. Eclipse Foundation. Eclipse Integrated Development Environment (IDE). Available at: https://www.eclipse.org, accessed 20.06.2018

[9]. Eclipse Foundation. Eclipse Java development tools (JDT). Available at: https://www.eclipse.org/jdt, accessed 20.06.2018

[10]. Klimov Yu.A. An approach to polyvariant binding time analysis for a stack-based language. First International Workshop on Metacomputation in Russia, Proceedings. Pereslavl-Zalessky, Russia, July 2–5, 2008. Pereslavl-Zalessky: Ailamazyan University of Pereslavl, 2008, pp. 78–84. Available at: http://meta2008.pereslavl.ru/accepted-papers/paper-info-6.html, accessed 20.06.2018

[11]. Klimov Yu.A. [Program specialization for object-oriented languages by partial evaluation: approaches and problems]. Preprinty` IPM im. M.V. Keldy`sha [Keldysh Institute Preprints], no. 12, 2008 (in Russian). Available at: http://library.keldysh.ru/preprint.asp?id=2008-12, accessed 20.06.2018

[12]. Klimov Yu.A. [Specializer CILPE: examples of object-oriented program specialization]. Preprinty` IPM im. M.V. Keldy`sha [Keldysh Institute Preprints], no. 30, 2008 (in Russian). Available at: http://library.keldysh.ru/preprint.asp?id=2008-30, accessed 20.06.2018

[13]. Klimov Yu.A. [SOOL: an object-oriented stacked-based language for specification and implementation of program specialization techniques]. Preprinty` IPM im. M.V. Keldy`sha [Keldysh Institute Preprints], no. 44, 2008 (in Russian). Available at: http://library.keldysh.ru/preprint.asp?id=2008-44, accessed 20.06.2018

[14]. Klimov Yu.A. [Specializer CILPE: binding time analysis]. Preprinty` IPM im. M.V. Keldy`sha [Keldysh Institute Preprints], no. 7, 2009 (in Russian). Available at: http://library.keldysh.ru/preprint.asp?id=2009-07, accessed 20.06.2018

[15]. Klimov Yu.A. [Specializer CILPE: residual program generation]. Preprinty` IPM im. M.V. Keldy`sha [Keldysh Institute Preprints], no. 8, 2009 (in Russian). Available at: http://library.keldysh.ru/preprint.asp?id=2009-08, accessed 20.06.2018

[16]. Klimov Yu.A. [Specializer CILPE: correctness proof]. Preprinty` IPM im. M.V. Keldy`sha [Keldysh Institute Preprints], no. 33, 2009, (in Russian). Available at: http://library.keldysh.ru/preprint.asp?id=2009-33, accssed 20.06.2018

[17]. Klimov Yu.A. [Specialization of programs in object-oriented languages by partial evaluation]. Ph.D. dissertation, Keldysh Institute of Applied Mathematics of RAS, Moscow, Russia, Nov 2009, 183 p. (in Russian). Available at: http://pat.keldysh.ru/~yura/publications/2009.10-Klimov-Disser-Specializacia_programm_na_ob'ektno-orientirovannyx_yazykah.pdf, accessed 20.06.2018

[18]. Klimov Yu.A. [Specializer CILPE: Partial evaluation for object-oriented languages]. Programmny`e sistemy`: teoriia i prilozheniia [Program Systems: Theory and Applications], no. 3(3), pp. 13–36, 2010 (in Russian). Available at: http://psta.psiras.ru/read/psta2010_3_13-36.pdf, accessed 20.06.2018

[19]. Bulyonkov M.A. and Kochetov D.V. Practical aspects of specialization of Algol-like programs. Dagstuhl Seminar on Partial Evaluation, Lecture Notes in Computer Science, O. Danvy, R. Gluck, and P. Thiemann, (Eds.), vol. 1110. Springer, 1996, pp. 17–32
[20]. Ershov A.P. and Itkin V.E. Correctness of mixed computation in Algol-like programs. MFCS, Lecture Notes in Computer Science, J. Gruska, (Ed.), vol. 53. Springer, 1977, pp. 59–77
[21]. Andersen L.O. Program analysis and specialization for the C programming language. Ph.D. dissertation, DIKU, University of Copenhagen, May 1994, (DIKU report 94/19)
[22]. Andersen L.O. Binding-time analysis and the taming of C pointers. Proceedings of the 1993 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM '93). ACM, 1993, pp. 47-58. Available at: http://dx.doi.org/10.1145/154630.154636, accessed: 20.06.2018
[23]. Consel C., Lawall J.L., and Meur A.-F.L. A tour of Tempo: a program specializer for the C language. Sci. Comput. Program., vol. 52, no. 1-3, 2004, pp. 341–370
[24]. Meur A.L., Lawall J.L. and Consel C. Towards bridging the gap between programming languages and partial evaluation. Proceedings of the 2002 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM '02), Portland, Oregon, USA, January 14-15, 2002, P. Thiemann, (Ed.). ACM, 2002, pp. 9–18. Available at: http://doi.acm.org/10.1145/503032.503033, accessed 20.06.2018
[25]. Schultz U.P., Lawall J.L. and Consel C. Automatic program specialization for Java. ACM Trans. Program. Lang. Syst., vol. 25, no. 4, 2003, pp. 452–499
[26]. Muller G., Moura B., Bellard F. and Consel C. Harissa: A flexible and efficient Java environment mixing bytecode and compiled code. Proceedings of the Third USENIX Conference on Object-Oriented Technologies (COOTS), June 16-20, 1997, Portland, Oregon, USA, S. Vinoski, (Ed.). USENIX, 1997, pp. 1–20. Available at: http://www.usenix.org/publications/library/proceedings/coots97/muller.html, accessed 20.06.2018.
[27]. Shali A. and Cook W.R. Hybrid partial evaluation. SIGPLAN Not., vol. 46, no. 10, Oct. 2011, pp. 375–390. Available at: http://doi.acm.org/10.1145/2076021.2048098, accessed 20.06.2018.
[28]. Ji R. and Bubel R. PE-KeY: A partial evaluator for Java programs. Proceedings of the 9th International Conference on Integrated Formal Methods, IFM'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 283– 295. Available at: http://dx.doi.org/10.1007/978-3-642-30729-4_20, accessed 20.06.2018
[29]. Ahrendt W., Beckert B., Bubel R., Hahnle R., Schmitt P.H. and Ulbrich M., (Eds.). Deductive Software Verification – The KeY Book – From Theory to Practice. Lecture Notes in Computer Science. Springer, 2016, vol. 10001. Available at: https://doi.org/10.1007/978-3-319-49812-6, accessed 20.06.2018
[30]. Würthinger T., Wimmer C., Wöß A., Stadler L., Duboscq G., Humer C., Richards G., Simon D., and Wolczko M. One VM to rule them all. Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, Onward! 2013. New York, NY, USA: ACM, 2013, pp. 187–204. Available at: http://doi.acm.org/10.1145/2509578.2509581, accessed 20.06.2018
[31]. Würthinger T., Wimmer C., Humer C., Wöß A., Stadler L., Seaton C., Duboscq G., Simon D., and Grimmer M. Practical partial evaluation for high-performance dynamic language runtimes. SIGPLAN Not., vol. 52, no. 6, Jun. 2017, pp. 662–676. Available at: http://doi.acm.org/10.1145/3140587.3062381, accessed 20.06.2018.

Интерактивный специализатор подмножества языка Java, основанный на методе частичных вычислений

[1] *И.А. Адамович <i.a.adamovich@gmail.com>*
[2] *Анд.В. Климов <klimov@keldysh.ru>*
[1] *Институт программных систем им. А.К. Айламазяна РАН,
152021, Россия, Ярославская обл., с. Веськово, ул. Петра Первого, д. 4а*
[2] *Институт прикладной математики им. М.В. Келдыша РАН,
125047, Россия, Москва, Миусская пл., д. 4*

Аннотация. Специализация — это оптимизация программ на основе использования наперёд заданной информации о значении части переменных. Методы специализации программ развиваются с 1970-х годов (смешанные вычисления, частичные вычисления, суперкомпиляция). Однако удивительно, что после трёх десятилетий разработанные специализаторы до сих пор не достигли того уровня, когда они станут пригодны для широкого практического применения. Возникает вопрос: в чём же причина? Наша гипотеза состоит в том, что задача специализации требуют гораздо большего участия человека в управлении процессом специализации, анализе результатов, проведении компьютерных экспериментов, чем в случае обычной оптимизации программы в компиляторах. Требуется погружение специализаторов в привычные для программистов интегрированные среды разработки, включая создание соответствующих диалоговых средств. В данной статье описываются результаты разработки и реализации методов интерактивной специализации на основе частичных вычислений для подмножества языка Java. Реализация выполнена в рамках популярной среды разработки (IDE) Eclipse. Разрабатываются сценарии человеко-машинного диалога с подсистемой специализации, интерактивные средства для составления задания на специализацию и управление процессом специализации. Приводится пример успешного применения разработанного специализатора. Остаточная программа работает в несколько раз быстрее чем исходная.

Ключевые слова: анализ программ; преобразование программ; интерактивная специализация программ; частичные вычисления; объектно-ориентированный язык; среда разработки программ

DOI: 10.15514/ISPRAS-2018-30(4)-2

Для цитирования: Адамович И.А., Климов Анд.В. Интерактивный специализатор подмножества языка Java основанный на методе частичных вычислений. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 29-44 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(4)-2

Список литературы

[1]. Jones N.D., Sestoft P. and Søndergaard H. An experiment in partial evaluation: the generation of a compiler generator. Rewriting Techniques and Applications, Lecture Notes in Computer Science, J.-P. Jouannaud, (Ed.), vol. 202. Springer-Verlag, 1985, pp. 124–140

[2]. Jones N.D., Gomard C.K., and Sestoft P. Partial Evaluation and Automatic Program Generation. Prentice-Hall, 1993. Доступно по ссылке: http://www.itu.dk/~sestoft/pebook/pebook.html, дата обращения: 20.06.2018

[3]. Futamura Y. Partial evaluation of computation process — an approach to a compiler-compiler. Systems, Computers, Controls, vol. 2, no. 5, 1971, pp. 45–50

[4]. Futamura Y. Partial evaluation of computation process — an approach to a compiler-compiler. Higher-Order and Symbolic Computation, vol. 12, no. 4, Dec 1999, pp. 381–391. Updated and revised version of [3]. Доступно по ссылке: http://doi.org/10.1023/A:1010095604496, дата обращения: 20.06.2018

[5]. Futamura Y. EL1 Partial Evaluator (Progress Report). Center for Research in Computing Technology, Harvard University, Tech. Rep., 1973. Доступно по ссылке: http://fi.ftmr.info/PE-Museum/EL1.PDF, дата обращения: 20.06.2018

[6]. Turchin V.F. The concept of a supercompiler. ACM Transactions on Programming Languages and Systems, vol. 8, no. 3, 1986, pp. 292–325

[7]. Turchin V.F. Supercompilation: techniques and results. Perspectives of System Informatics, Second International Andrei Ershov Memorial Conference, Akademgorodok, Novosibirsk, Russia, June 25-28, 1996. Proceedings, Lecture Notes in Computer Science, D. Bjørner, M. Broy, and I.V. Pottosin, (Eds.), vol. 1181. Springer, 1996, pp. 227–248

[8]. Eclipse Foundation. Eclipse Integrated Development Environment (IDE). Доступно по ссылке: https://www.eclipse.org, дата обращения: 20.06.2018

[9]. Eclipse Foundation. Eclipse Java development tools (JDT). Доступно по ссылке: https://www.eclipse.org/jdt, дата обращения: 20.06.2018

[10]. Klimov Yu.A. An approach to polyvariant binding time analysis for a stack-based language. First International Workshop on Metacomputation in Russia, Proceedings. Pereslavl-Zalessky, Russia, July 2–5, 2008. Pereslavl-Zalessky: Ailamazyan University of Pereslavl, 2008, pp. 78–84. Доступно по ссылке: http://meta2008.pereslavl.ru/accepted-papers/paper-info-6.html, дата обращения: 20.06.2018

[11]. Климов Ю.А. Особенности применения метода частичных вычислений к специализации программ на объектно-ориентированных языках. Препринты ИПМ им. М.В. Келдыша, № 12, 2008. Доступно по ссылке: http://library.keldysh.ru/preprint.asp?id=2008-12, дата обращения: 20.06.2018

[12]. Климов Ю.А. Возможности специализатора CILPE и примеры его применения к программам на объектно-ориентированных языках. Препринты ИПМ им. М.В. Келдыша, № 30, 2008. Доступно по ссылке: http://library.keldysh.ru/preprint.asp?id=2008-30, дата обращения: 20.06.2018

[13]. Климов Ю.А. SOOL: объектно-ориентированный стековый язык для формального описания и реализации методов специализации программ. Препринты ИПМ им. М.В. Келдыша, № 44, 2008. Доступно по ссылке: http://library.keldysh.ru/preprint.asp?id=2008-44, дата обращения: 20.06.2018

[14]. Климов Ю.А. Специализатор CILPE: анализ времен связывания. Препринты ИПМ им. М.В. Келдыша, № 7, 2009. Доступно по ссылке: http://library.keldysh.ru/preprint.asp?id=2009-07, дата обращения: 20.06.2018

[15]. Климов Ю.А. Специализатор CILPE: генерация остаточной программы. Препринты ИПМ им. М.В. Келдыша, № 8, 2009. Доступно по ссылке: http://library.keldysh.ru/preprint.asp?id=2009-08, дата обращения: 20.06.2018

[16]. Климов Ю.А. Специализатор CILPE: доказательство корректности. Препринты ИПМ им. М.В. Келдыша, № 33, 2009. Доступно по ссылке: http://library.keldysh.ru/preprint.asp?id=2009-33, дата обращения: 20.06.2018

[17]. Климов Ю.А. Специализация программ на объектно-ориентированных языках методом частичных вычислений. Дис. к.ф.-м.н., Институт прикладной математики им. М.В. Келдыша РАН, Москва, Россия, ноябрь 2009, 183 стр. Доступно по ссылке: http://pat.keldysh.ru/~yura/publications/2009.10-Klimov-Disser-Specializacia_programm_na_ob'ektno-orientirovannyx_yazykah.pdf, дата обращения: 20.06.2018

[18]. Климов Ю.А. Специализатор CILPE: частичные вычисления для объектно-ориентированных языков. Программные системы теория и приложения, № 3(3), 2010, стр. 13–36 Доступно по ссылке: http://psta.psiras.ru/read/psta2010_3_13-36.pdf, дата обращения: 20.06.2018

[19]. Bulyonkov M.A. and Kochetov D.V. Practical aspects of specialization of Algol-like programs. Dagstuhl Seminar on Partial Evaluation, Lecture Notes in Computer Science, O. Danvy, R. Gluck, and P. Thiemann, (Eds.), vol. 1110. Springer, 1996, pp. 17–32

[20]. Ershov A.P. and Itkin V.E. Correctness of mixed computation in Algol-like programs. MFCS, Lecture Notes in Computer Science, J. Gruska, (Ed.), vol. 53. Springer, 1977, pp. 59–77

[21]. Andersen L.O. Program analysis and specialization for the C programming language. Ph.D. dissertation, DIKU, University of Copenhagen, May 1994, (DIKU report 94/19)

[22]. Andersen L.O. Binding-time analysis and the taming of C pointers. Proceedings of the 1993 ACM SIGPLAN symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM '93). ACM, 1993, pp. 47-58. Доступно по ссылке: http://dx.doi.org/10.1145/154630.154636, дата обращения: 20.06.2018

[23]. Consel C., Lawall J.L., and Meur A.-F.L. A tour of Tempo: a program specializer for the C language. Sci. Comput. Program., vol. 52, no. 1-3, 2004, pp. 341–370

[24]. Meur A.L., Lawall J.L. and Consel C. Towards bridging the gap between programming languages and partial evaluation. Proceedings of the 2002 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM ’02), Portland, Oregon, USA, January 14-15, 2002, P. Thiemann, (Ed.). ACM, 2002, pp. 9–18. Доступно по ссылке: http://doi.acm.org/10.1145/503032.503033, дата обращения: 20.06.2018

[25]. Schultz U.P., Lawall J.L. and Consel C. Automatic program specialization for Java. ACM Trans. Program. Lang. Syst., vol. 25, no. 4, 2003, pp. 452–499

[26]. Muller G., Moura B., Bellard F. and Consel C. Harissa: A flexible and efficient Java environment mixing bytecode and compiled code. Proceedings of the Third USENIX Conference on Object-Oriented Technologies (COOTS), June 16-20, 1997, Portland, Oregon, USA, S. Vinoski, (Ed.). USENIX, 1997, pp. 1–20. Доступно по ссылке: http://www.usenix.org/publications/library/proceedings/coots97/muller.html, дата обращения: 20.06.2018

[27]. Shali A. and Cook W.R. Hybrid partial evaluation. SIGPLAN Not., vol. 46, no. 10, Oct. 2011, pp. 375–390. Доступно по ссылке: http://doi.acm.org/10.1145/2076021.2048098, дата обращения: 20.06.2018

[28]. Ji R. and Bubel R. PE-KeY: A partial evaluator for Java programs. Proceedings of the 9th International Conference on Integrated Formal Methods, IFM’12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 283– 295. Доступно по ссылке: http://dx.doi.org/10.1007/978-3-642-30729-4_20, дата обращения: 20.06.2018

[29]. Ahrendt W., Beckert B., Bubel R., Hahnle R., Schmitt P.H. and Ulbrich M., (Eds.). Deductive Software Verification — The KeY Book — From Theory to Practice. Lecture Notes in Computer Science. Springer, 2016, vol. 10001. Доступно по ссылке: https://doi.org/10.1007/978-3-319-49812-6, дата обращения: 20.06.2018

[30]. Würthinger T., Wimmer C., Wöß A., Stadler L., Duboscq G., Humer C., Richards G., Simon D., and Wolczko M. One VM to rule them all. Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, Onward! 2013. New York, NY, USA: ACM, 2013, pp. 187–204. Доступно по ссылке: http://doi.acm.org/10.1145/2509578.2509581, дата обращения: 20.06.2018

[31]. Würthinger T., Wimmer C., Humer C., Wöß A., Stadler L., Seaton C., Duboscq G., Simon D., and Grimmer M. Practical partial evaluation for high-performance dynamic language runtimes. SIGPLAN Not., vol. 52, no. 6, Jun. 2017, pp. 662–676. Доступно по ссылке: http://doi.acm.org/10.1145/3140587.3062381, дата обращения: 20.06.2018

Heterogeneous Architectures Programming Library

G.V. Kirgizov <gkirgizov@gmail.com>
I.A. Kirilenko <y.kirilenko@spbu.ru>
Software Engineering Department,
Saint Petersburg State University,
University Embankment, 7, Saint Petersburg, 199034, Russia.

Abstract. Embedded platforms with heterogeneous architecture, considered in this paper, consist of one primary and one or more secondary processors. Development of software systems for these platforms poses substantial difficulties, requiring a distinct set of tools for each constituent of the heterogeneous system. It also makes achieving high efficiency the more difficult task. Moreover, many use cases of embedded systems require runtime configuration, that cannot be easily achieved with usual approaches. This work presents a C-like metaprogramming DSL and a library that provides a unified interface for programming secondary processors of heterogeneous systems with this DSL. Together they help to resolve aforementioned problems. The DSL is embedded in C++ and allows to freely manipulate its expressions and thus embodies the idea of generative programming, when the expressive power of high-level C++ language is used to compose pieces of low-level DSL code. Together with other features, such as generic DSL functions, it makes the DSL a flexible and powerful tool for dynamic code generation. The approach behind the library is dynamic compilation: the DSL is translated to LLVM IR and then compiled to native executable code at runtime. It opens a possibility of dynamic code optimizations, e.g. runtime function specialization for specific parameters known only at runtime. Flexible library architecture allows simple extensibility to any target platform supported by LLVM. At the end of the paper a system approbation on different platforms and a demonstration of dynamic optimizations capability are presented.

Keywords: metaprogramming; code generation; embedded DSL; heterogeneous systems; embedded systems.

DOI: 10.15514/ISPRAS-2018-30(4)-3

For citation: Kirgizov G.V., Kirilenko I.A. Heterogeneous Architectures Programming Library. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 4, 2018, pp. 45-62. DOI: 10.15514/ISPRAS-2018-30(3)-3

1. Introduction

Embedded systems have been in a widespread use a long time, and today they become even more relevant because of the rapid development and adoption of new application fields, for example, Internet-of-Things, ”smart houses” and robotics.

Many of the embedded systems used in these areas have heterogeneous architectures due to nature of their tasks. Typically, they consist of one primary, more powerful processor that executes the main program and performs common control, and one or several secondary microcontrollers or processors that provide read/write access to sensors and peripheral devices or may perform some other special functions. Examples of such systems may be: Raspberry Pi (main) + Arduino with Atmel AVR (peripheral) and Odroid XU4 (main) + stm32f4 microcontroller (peripheral).

Heterogeneity of these systems causes noticeable overhead. Traditional development workflow requires use of IDEs and toolchains that are specific for each part of the system. This need to develop each part of the system in a separate project using a different set of platform-specific tools makes system development processes more complex and expensive. The amounts of resources required for support and changes also grow.

The efficiency of the system suffers too. Due to specificities of each microcontroller and their limited hardware capabilities they often have only basic firmware, which only capabilities are reading sensors, communicating results back to main processor, receiving data and control commands from it and writing the received data to special registers of peripheral devices. All core program logic is contained on the primary processor, and, as secondary processors/microcontrollers do not contain even a part of this logic, constant communication between them is unavoidable (because of the nature of control cycle: request sensor data, wait for it to arrive, compute control output, send it back to the secondary processors, repeat).

This work is based on preliminary results of [1] that showed the viability of the idea of dynamic code generation. We revise previous architectural choices, fully reimplement the library because of shortcomings of existing implementation and substantially extend it in terms of functionality and possible applications/uses.

In particular, the new DSL is completely abstracted from other parts of the library and can be used independently in other projects based on the idea of metaprogramming. Moreover, the new DSL implementation allows employing various dynamic optimizations, which are not possible in heterogeneous systems using traditional programming techniques. The contribution of this work is twofold. We present:

- C++ embedded DSL for dynamic metaprogramming;
- a library that simplifies development of programs for heterogeneous systems providing unified programming interface; it also allows to achieve higher efficiency of the system and implement better organizations of work between its parts.

The library is based on the idea of a dynamic compilation of programs for peripheral processors.

We also demonstrate system’s capabilities on a number of examples that show important features of the new DSL and some applications in embedded systems domain. Source code with build instructions can be found in the project repository[1].

Several possible use cases of this library can be imagined. First use case is avoiding the overhead of constant communication between processors. Of course, it’s possible to accomplish it without this library: move part of the program logic to peripheral processors on top of their basic firmware. However, with usual tools, it incurs additional costs for development and support because with this approach there is no more single point of change in core logic of the system. There is unavoidable need to support several projects and ensure proper integration. Whereas presented library allows avoiding both communication overhead and unnecessary complexity of the development process.

The second use case is to allow dynamic specialization of heterogeneous systems for their operating environment. Some types of embedded heterogeneous systems can be deployed in a wide range of environments with various conditions. When their operation depends on these conditions, developers of programs for such systems must anticipate in the code all possible conditions. It may be implemented through constant monitoring of the environment. Another alternative is on-place configuration or tuning of each particular system. However, it may not be possible due to nature of the task or too often or rapid (for manual operating) changes of the environment. Another variation of dynamic specialization scenario is a runtime configuration for specific peripheral devices (e.g. different models of sensors and actuators).

Our library can help there in the case of sufficiently slowly changing environment (relative to a number of control cycles, when the time required for dynamic recompilation will pay off). It can be better shown on the specific example of PID controller tuning. Firstly, PID controller with tuning subpro gram is loaded on the peripheral microcontroller. Then, when optimal parameters are found, microcontroller program can be recompiled with these particular coefficients, thus yielding system that is maximally suited for its operating conditions. For the specific case of not changing environment this tuning and dynamic recompilation can be executed only once on deployment. This example is elaborated on in greater detail in the section Demonstration.

The paper is organized as follows. The next section discusses similar works that are based on the similar ideas. The third section describes main architectural decisions and presents the architecture of the system. The fourth section is devoted to the DSL and provides a reader with a number of examples. The following section describes other parts of the system and their functionality in greater detail. The Approbation section describes test setups and the Demonstration section shows benefits of

[1]https://github.com/gkirgizov/hetarch

dynamic recompilation on a specific example and discusses scope and applicability of the library. The paper is closed with conclusion and discussion of possible directions of further work.

2. Similar Work

The difficulties, which heterogeneous systems cause, are not unique for the embedded software engineering. Programming of heterogeneous systems is an old problem, and there are several conceptual approaches to aforementioned difficulties.

The most known area that faces it is programming with graphical processors. In this case, heterogeneous system consists of CPU and one or more GPUs. (The case of graphics programming, i.e. using shaders and graphics pipelines, is further from heterogeneous programming and is not considered here.) It is an old problem in this field: how to effectively and, not less importantly, conveniently use GPU in usual, CPU-centric programs? There are two main examples of systems that answer this question: Open Computing Language (OpenCL) [2] and CUDA framework from Nvidia [3]. Both these frameworks propose the use of C and C++ languages extended with special functions and attributes for writing device code (code to be executed on secondary processors). It can be written, depending on user’s aims and requirements, either in separate files or in the main program files together with usual C/C++ host code that is intended to be executed on CPU. OpenCL uses dynamic compilation (at runtime) of device code; some device vendors provide offline compilers for their devices (for example, Intel Code Builder for OpenCL API). CUDA similarly provides both possibilities: Nvidia has an offline compiler called NVCC and a runtime compilation library NVRTC.

The motivation behind these examples and presented in this paper library is essentially the same: use of the same programming interface for all constituents of a heterogeneous system.

Another area that this work touches is the ideas of generative, multi-stage programming and runtime code generation. A good discussion of general motivations and trade-offs behind these ideas, as well as examples of some actual realizations and a number of references provides [4].

Among their examples Delite—a heterogeneous parallel framework for domain-specific languages [5], [6]—is of particular interest. Delite’s focus is on the performance of parallel heterogeneous systems, e.g. mixed CPU/GPU architectures and clusters. It is built on top of Lightweight Modular Staging (LMS) [7] system, that makes use of a form of metaprogramming to construct a symbolic representation of a DSL program. LMS provides a basis for DSLs embedded in Scala. On top of this layer, Delite is structured into a compiler framework and a runtime component. The framework provides primitives for parallel operations and generates Scala, CUDA or C++ code from DSLs.

Although both we and the authors of Delite start from the same idea of multi-stage programming, our systems significantly differ in the approaches and application domains. Most importantly, we use dynamic code generation and thus employ the generative programming at runtime to achieve dynamic optimizations. The authors of Delite, on the other hand, require static compilation of DSLs—they promote the use of additional compilation stage to perform domain-specific optimizations.

3. High Level Description

Further in the text by the word host is meant primary processor, by target—one of the peripheral processors or microcontrollers, by the user—developer who uses this library.

3.1 Main Architectural Decisions

The following decisions have shown themselves as reasonable and grounded and thus are inherited from the previous work [1]. They are discussed here to provide better context.

Runtime changes in executable code on targets can be achieved by two approaches: dynamic compilation, which happens on the host, and code interpretation which happens on targets. Because modern interpreted languages generally have higher requirements and cause more overhead, the first decision is to use dynamic compilation on the more powerful host.

The second decision is to use embedded domain specific language (DSL) as a basis for dynamic code generation. An alternative of using code attributes with compiler extension (e.g. as used by OpenCL) is less viable due to several reasons. First, code defined in a such way can be manipulated at the runtime only as a string of characters. It complicates analysis and dynamic code specialization, requiring additional step of semantic analysis before that, whereas DSL approach gives semantic information ’for free’. Second, it is more demanding to maintain the compiler extension to keep it up-to-date with the needed compiler versions. In addition, it is still necessary to use dynamic compilation tools. It seems excessive to support both the compiler extension and the dynamic compilation tools. Moreover, it would restrict library users to only one compiler, which can be especially inconvenient in the world of embedded systems.

LLVM [8] is used as a compilation backend. There is no real alternative, and its excellent design and convenience of use made this work possible.

C++ is chosen as a language of implementation by several reasons: firstly, it is a natural choice for embedded systems domain; secondly, it allows to avoid overhead of interfacing with LLVM; and, most importantly, with template metaprogramming it provides the necessary expressive power for implementation of the DSL, which itself must be very expressive and general to be applicable in a wide range of use cases. Specifically, the latest C++17 standard is used.

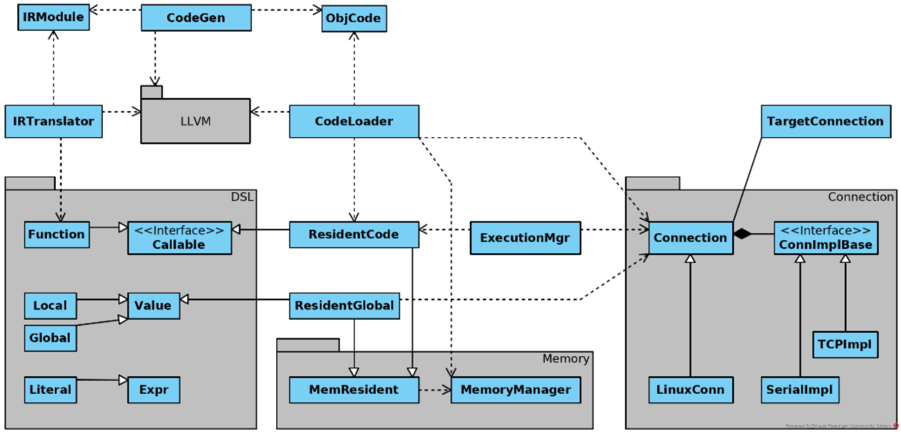


Fig. 1. UML class diagram of the system. DSL class hierarchy is shown only approximately because of its breadth and dynamic nature. IRTranslator together with non-resident DSL constructs constitute independent and reusable DSL subsystem.

3.2 Architecture Overview

DSL allows the user to describe the code, which will be executed on targets. CodeGen module provides a simplified interface to LLVM compilation and optimization facilities. CodeLoader, Execution and Connection modules let user load code on targets, communicate with them (for example, using global variables) and control the code execution. Management of the target's memory is provided by the host through MemoryManager module.

Fig. 1 shows the structure of the system.

This architecture has a benefit of simple extensibility. Each of the following parts of the library can be extended independently from others:

- DSL constructs and operations (for example, support array slicing or exponentiation at the language level);
- communication protocols;
- target runtime functionality;
- most importantly, target platforms.

For details on these points, the reader can proceed to the following sections.

4. DSL

4.1 Design

The core of this library is a powerful embedded C-like DSL. It is translated to LLVM Intermediate Representation (IR) to allow code compilation for a wide range of targets supported by LLVM. This design of the DSL as translated and compiled at runtime is directly motivated by the concept of generative (or multi-stage) programming when the abstraction power of high-level languages is used to compose pieces of low-level code [4]. It makes runtime code generation and domain-specific optimization a fundamental part of the program logic.

As authors of [4] note, the usual appeal of DSLs is in increasing productivity by providing a higher level, more intuitive programming model for domain experts, who are not necessarily expert programmers ("user-facing" DSLs). The other direction, which is of interest for us in this paper, is in using DSL as a means for exposing knowledge about high level program structures to a compiler.

This DSL implementation makes heavy use of powerful template metaprogramming capabilities of C++, up to C++17 standard. The idea to leverage C++ templates to cope with challenges that poses development of DSLs aimed at generative programming goes back at least to the work of Czarnecki et al. [9].

4.2 Description and Examples

DSL provides all necessary language constructs with a familiar syntax:

- basic types (possibly cv-qualified):
  - arithmetic types;
  - pointers;
  - arrays of fixed length (possibly nested);
  - structs (possibly nested);
- operations:
  - arithmetic operators (with the support of pointer arithmetic);
  - logical operators;
  - bitwise operators;
  - C-like cast;
- control flow expressions:
  - sequential (comma operator expression);
  - conditional (if-else expression);
  - while loop;
- functions (with a fixed number of arguments; no recursion);
- literal values.

It is also easily extensible with other higher-level constructs (for example, Python-like array slicing) which will be translated directly to LLVM IR (i.e. will be efficient).

To allow simpler organization of the language, every DSL construct models either value or expression; there are no statements. For example, to return void from a function user needs to use special DSL construct '`Unit`'. Loops naturally return value from their last cycle. If loop did not run it returns default-initialized value (generally, zero-initialized).

Any DSL construct has a corresponding underlying C++ type, which determines allowed operations on it and conversions to other types. Underlying C++ type can be accessed through member type alias `::type` which is present in every DSL type. And the DSL value type can be obtained (if there is one) from C++ type using `to_dsl<T>` type trait. In other words, there is a direct mapping between DSL types and C++ types. Type trait `to_dsl<T>` can be used as a convenient type factory.

Type of the DSL constructs (real C++ type, not the underlying C++ type) encodes how it was constructed and what child DSL constructs constitute it (for example see listing 1).

```
Var<int> x, y, z;
auto expr = (x + y) * z;
using expr_type =
  EBinOp< Instruction::FMul,
          EBinOp< Instruction::Add,
                  Var<int>,
                  Var<int>
          >,
          Var<int>
  >;
```

Listing 1. Type of some DSL expression

One of the most interesting features of the DSL is a separation of DSL abstract syntax tree (AST) construction from DSL function instantiation. It is achieved through the use of C++14 generic lambdas which play a role of DSL code generators (AST builders). Example can be seen on the next listing.

```
auto max_gen = [](auto x, auto y) {
    return If(x > y, x, y);
};
auto dsl_max = make_dsl_fun<int, int>(max_gen);
```

It allows simple and effective reuse of needed DSL constructs, as in the next example.

```
auto max3_gen = [&](auto x1, auto x2, auto x3) {
    return max_gen(x1, max_gen(x2, x3));
};
auto dsl_max3 = make_dsl_fun<int, int, int>(max3_gen);
```

This conceptually differs from simple function call as a means of code reuse and is closer to function inlining. In this way the new DSL generator is constructed which, in its turn, can be later reused. Moreover, on the point of DSL code generation user can utilize C++ constructs to build more complex DSL expressions (Listing 2).

```
// note: accepts arbitrary DSL expressions
auto reduce_sum_gen = [](auto ...xs) {
    // Using C++17 fold expression
    return (... + xs);
};
auto sum3 = make_dsl_fun<float, double, int>(reduce_sum_gen);
```

Listing 2. Use C++ code to build complex DSL expressions.

```
// note: accepts arbitrary DSL expressions
//  (e.g. other generators)
auto get_reducer = [](const auto& binary_op) {
    return [&](auto x1, auto... xs) {
        // Using C++17 fold expression
        return ( (x1 = binary_op(x1, xs)), ... );
        // Redundant assignments
        //  will be optimized out by LLVM
    };
};
auto max_vararg_gen = get_reducer(max_gen);
auto max3 = make_dsl_fun<int, int, int>(max_vararg_gen);
```

Listing 3. Generator of DSL reduce function over arbitrary DSL expressions.

Listing 4 shows two noticeable syntactic features of the DSL: the sequential operator that plays a role of C/C++ semicolon and DSL local variables. Generally, any DSL variable which is not an argument of DSL generator (enclosing lambda) will be considered a local one. For the more consistent syntax user can define local variables inside the generator lambdas. Also, note that they can't be defined inside the DSL expressions because they follow the rules of C++ expressions. To use global variables a user is required to first load them on the target because they are translated to LLVM IR as actual memory addresses.

```
Var<int> local1;
 // note lambda capture (can also be [&])
auto max_gen = [=](auto arg) {
    Var<int> local2;
    return (
        // variables can't be defined here!
        local1 += arg,
        local1 += local2,
        arg // last expression is returned
    );
};
```

Listing 4. Use of comma operator and local variables.

The next listing demonstrates that DSL allows to construct complex expressions in familiar, close to C, syntax.

```
auto complex_expr = [](Ptr<Var<uint32_t>> ptr) {
    Var<uint32_t> tmp;
    return tmp = *ptr &= ~(*++ptr ^ Lit(1 << 8));
};
```

Generic DSL functions is another very useful feature. As can be seen from previous examples, DSL generators are not bound to specific types of parameters. Instead of explicit manual instantiation of DSL function with required types of parameters library user can instantiate generic DSL function with a help of function factory. If generic function is used with arguments of inappropriate types, compiler will catch this and compilation will fail with comprehensible error message.

Instantiated generic functions are stored in a function repository by a key which represents their type. As a type of DSL constructs encodes their AST, type of DSL functions encodes their body. Thus, the structural equivalence between functions is achieved without any overhead. Thanks to this repeated instantiation of the (structurally) same DSL functions is avoided. DSL function is deleted from the repository at the end of translation to LLVM IR. Needless to say, all this happens behind the scenes and a user isn't required to know about these details.

Listing 5 shows an example of the use of a generic DSL function.

```
auto generic_max = make_generic_dsl_fun(max_gen);

auto max4_gen = [&](auto x1, auto x2, auto x3, auto x4) {
    return generic_max(
            generic_max(x1, x2),
            Cast<float>(generic_max(x3, x4))
    );
};
// This will cause instantiation of 2 max functions:
//  for ints and for floats
auto max4 = make_dsl_fun<float, float, int, int>(max4_gen);
```

Listing 5. Generic DSL function example

Last, but not the least, DSL is designed with usability in mind. C++ code with a heavy use of templates is known for its complex error message on compilation failure. In DSL all major type constraints are checked with static assert standard library function which produces comprehensible compile time error messages.

5. Subsystems Description

5.1 MemoryManager

This centralized memory management organization allows to free less powerful targets from extra tasks and avoid extra communication cycles which would be inevitable to ensure correct memory allocation if targets managed their memory themselves. Best-fit, worst-fit and first-fit memory management algorithms are implemented. Conceptually MemoryManager is part of a CodeLoader and used only for data and code loading. That is, it's important to note that target code can't dynamically allocate memory on targets.

5.2 CodeLoader

With the help of CodeLoader module user can load DSL global variables and compiled code on targets. CodeLoader also allows getting a handle to already loaded variables and functions. In this case, no checks or memory allocation is performed, because, in general, there is no possibility to ensure correctness of user's actions. For example, functions can be loaded on a target in a persistent memory in one program run, and on another program run any knowledge about it will be lost, whereas the user may want to access previously loaded data and functions. So, it is assumed that user knows what he is doing.

5.3 Connection Module: Host side

Connection module consists of two parts: command protocol for communication between host and targets and underlying connection implementation. The functionality of the former is fully built on the primitives of the latter, which must provide synchronous read and write operations.

The core command protocol includes the following commands:

- echo (for testing);
- read specified number of bytes at a specified address;
- write data to a specified address;
- call function at the specified address (without arguments and return value);
- set function at the specified address on execution by the timer;
- set function at the specified address on execution on the specific interrupt.

This abstraction from specific implementation allows easier extensibility on new connection protocols. This work implements connection through TCP and through USB (used as a virtual serial port).

5.4 Connection Module: Target Runtime API

Each specific target platform requires its own firmware to interface with the host. It must provide functionality for communicating with the host and answering to requests according to the command protocol.

At this point an important consideration arises: targets must provide API sufficient for a wide range of tasks. Generally, peripheral devices on microcontrollers are memory mapped, which means that runtime API consisting of memory read and write functions can be sufficient. For example, the family of STM32 microcontrollers has fixed memory map and each device has a specific predefined address in memory.

Some platforms may need an extended API. When the target has an operating system, in particular Linux, it can additionally provide an interface to some of the system calls: `open()` for using devices represented as input/output ports and `mmap()` for correct work with library runtime process address space. It is implemented in the LinuxConnection module. Although for this platform it is also possible to implement an interface to arbitrary system calls and libraries using `dlopen()` and `dlsym()` functionality, the library runtime API for Linux is intentionally left minimal but sufficient for tasks concerned with controlling peripheral devices.

Another important question is a debugging interface. Issuing diagnostic messages to some local to target buffer can accomodate most of the needs and at the same time is easily implementable. Target must provide interface to read the buffer and to get an address of the target local logging function. This address is used to construct the DSL wrapper for remote logging function. From this point it can be further used in the DSL code.

6. Approbation

The system was tested on several setups:

- Linux on x86 plays the role of both host and target machines, communication is through TCP connection (setup for tests during development);
- the host is Linux x86, the target is Odroid XU4 (armv7a) with Linux, TCP connection;
- the host is Linux x86, the target is bare-bones stm32f429i-discovery microcontroller (armv7em), USB Virtual COM Port connection;
- the host is Odroid XU4 (armv7a) with Linux, the target is bare-bones stm32f429i-discovery (armv7em), connection through USB Virtual COM Port.

Tests were performed for each command from the command protocol (see above in the section 5.3).

7. Demonstration

For a demonstration of dynamic optimization possibilities, which this library opens, the reader can refer to the following listings of PID control (listing 6) and its tuning (listing 7) for specific conditions of the deployment environment.

```
using namespace hetarch;
using namespace hetarch::dsl;

typedef int32_t ctrl_t; // for control variables
typedef float coef_t; // for coefficients

// Example of the target
typedef uint32_t addr_t; // size_t of the target
conn::SerialConnImpl<addr_t> conn{"/dev/ttyACM0"};
SimplePipeline<addr_t> pipeline{"armv7e_linux_eabihf", conn};

// Global var-s to store error data between control cycles
auto perr = pipeline.load(Global{ Var<ctrl_t>{0} });
auto ierr = pipeline.load(Global{ Var<ctrl_t>{0} });

// dt -- control cycle durations (in seconds)
// sp -- setpoint
auto pid_gen = [&](auto Kp,auto Ki,auto Kd,auto dt,auto sp) {
  auto pid_ctrl = [&]{
    // Local variables:
    // pv -- process variable
    // cv -- control variable
    Var<ctrl_t> pv, cv, prev_perr, derr;

    // read_pv and write_cv are some dsl generators
    //  that perform actual input/output
    return (
      pv = read_pv(),
```

```
            prev_perr = perr,
            perr = sp - pv,
            ierr += perr,
            derr = perr - prev_perr,
            cv = Kp*perr + Kd*derr/dt + Ki*ierr*dt;

            write_cv(cv)
          );
        };
        return pid_ctrl;
      };
```

Listing 6. PID controller DSL code.

```
auto tuner = [&](auto dt, auto sp){
  // For tuning coefficients are usual mutable DSL variables
  Var<coef_t> Kp{0}, Ki{0}, Kd{0};
  auto pid_ctrl = pid_gen(Kp, Ki, Kd, dt, sp);

  // Specific tuning method:
  //  determines current operating conditions
  //  (e.g. by reading some sensors)
  //  and returns tuning data that allows to compute
  //  optimal PID controller coefficients.
  // E.g. for Ziegler-Nichols method it is
  //  Ku -- "ultimate gain" and Tu -- oscillation period
  return (/* actual tuning code goes here */);
};

// Example parameters
Lit sp{42}; // Setpoint
int ms_delay{100}; // Control cycle duration
Lit dt{ms_delay / 1000.0};

auto tuning_code = make_dsl_fun(tuner, dt, sp);
// Translate, compile and load tuning code
auto tuning_fun = pipeline.load(tuning_code);
// Run tuning code and get tuning data
auto tuning_data = exec.call(tuning_fun, dt, sp);
// Compute coefficients using optimal tuning data
auto [Kp, Ki, Kd] = compute_coefs(tuning_data);

// Generate optimal PID controller
auto opt_pid_gen = pid_gen(Kp, Ki, Kd, dt, sp);
auto opt_pid_code = make_dsl_fun(opt_pid_gen);
// Translate, compile and load optimal PID controller
auto opt_pid = pipeline.load(opt_pid_dsl);

// Finally, run PID controller on timer
pipeline.schedule(opt_pid.callAddr, ms_delay);
```

Listing 7. PID tuning DSL code.

The work is organized in the following way:

- in the first phase host loads general version of the PID controller with tuning code on the target;
- in the second phase tuning code is called and its result is read by host;
- in the third phase host computes coefficients based on tuning data and recompiles PID controller with them;
- finally, host loads PID controller optimized for specific coefficients.

This example shows two advantages of using the library. Firstly, tuning code is completely absent from the final program running on the target. Dynamic code generation allows compiling code for specific constant coefficients to achieve better execution times and smaller program size.

Secondly, the dynamically generated code can be more optimal due to optimizations performed by LLVM. When coefficients are integer values, or, even better, integer powers of two (or float values, that can be rounded without big errors), resulting code will be generated with fewer (or completely without) expensive floating operations.

```c
typedef int ctrl_t;
typedef float coef_t;

extern coef_t Kp, Kd, Ki;
ctrl_t perr = 0, ierr = 0;

ctrl_t pid_ctrl(float dt, ctrl_t sp, ctrl_t pv) {
  ctrl_t prev_perr = perr;
  perr = sp - pv;
  ierr += perr;
  ctrl_t derr = perr – prev_perr;

  return Kp*perr + (Kd*derr/dt) + (Ki*ierr*dt);
}
```

Listing 8. PID controller C code used for LLVM IR comparison.

To emphasize possible dynamic optimizations, fig. 2 presents a comparison between listings of the PID controller code for two cases:

- C code from listing 8 compiled with clang without this library;
- DSL code from listing 6 dynamically optimized with this library.

```
1.  ; Kp * perr
2.  %9 = load float, float* @Kp
3.  %10 = sitofp i32 %perr to float
4.  %11 = fmul float %9, %10
5.
6.  ; Kd * derr / dt
7.  %12 = load float, float* @Kd
8.  %13 = sitofp i32 %derr to float
9.  %14 = fmul float %12, %13
10. %15 = fdiv float %14, %dt
11.
12. %16 = fadd float %11, %15
13.
14. ; Ki * ierr * dt
15. %17 = load float, float* @Ki
16. %18 = sitofp i32 %ierr to float
17. %19 = fmul float %17, %18
18. %20 = fmul float %19, %dt
19.
20. %21 = fadd float %16, %20
```

```
1.
2.
3.  ; Kp * perr
4.  %12 = shl i32 %perr, 2
5.  %13 = sitofp i32 %12 to float
6.
7.  ; Kd * derr / dt
8.  %14 = sitofp i32 %derr to float
9.  %15 = fmul float %14, 5.000000e-01
10. %16 = fdiv float %15, 1.000000e-01
11.
12. %17 = fadd float %16, %13
13.
14.
15. ; Ki * ierr * dt
16. %18 = mul i32 %ierr, 6
17. %19 = sitofp i32 %18 to float
18. %20 = fmul float %19, 1.000000e-01
19.
20. %21 = fadd float %20, %17
```

Fig. 2. Comparison of LLVM IR generated for expression "`Kp*perr + (Kd*derr/dt) + (Ki*ierr*dt)`" (core part of the PID controller code; other lines are omitted here). Compiler options used: `-O2 -target x86_64-pc-linux-gnu`. LLVM IR is used instead of native assembler because it is more readable and optimizations are done on the IR. Left: compiled with clang from C code on list. 8. LLVM IR is presented only for the last line. Right: compiled with LLVM from DSL (see list. 6). For the sake of demonstration it is assumed that dynamically determined PID controller coefficients are `Kp=4, Kd=6, Ki=0.5`; and control cycle duration is `dt=0.1`.

There are several things on the fig. 2 to note:

- dynamically generated code has fewer memory accesses because it is compiled for specific values (note lines 2, 7, 15 where usual code loads coefficients stored as global variables);
- instead of floating-point multiplications (lines 4 and 17 on the left) integer shift (line 4, right) and integer multiplication (line 16, right) are used;
- one apparent to a programmer optimization on line 9, right is missed: substitute multiplication by 0.5 with integer division by 2 or right shift by one; and it should be[2], although it is possible to implement such optimizations on the DSL level.

7.1 Library Applicability

The library is intended for use with embedded heterogeneous systems of a small scale with low-power secondary processors and microcontrollers that run heterogeneous tasks. The case of homogeneous tasks on the more powerful systems is better accomodated with existing tools (e.g. OpenCL or Delite) that are

[2]This compiler behavior is expected according to C11 standard (section F9.2.1), because representations of 0.5 and 2 maybe not be equivalent and the result can be different on some machines.

specifically aimed at scheduling and parallelizing the computations across bigger number of secondary processors. This library is not intended for such use cases and doesn't provide any orchestration for parallel tasks. Each secondary processor should be managed manually and separately.

Generally, the benefits and applicability of the library should be considered in each particular case. As noted in the introduction, the library is well suited for the problems when the dynamic configuration of the system is required (either for particular environment conditions or for different peripheral devices and sensors). It's also important to consider the price of dynamic recompilation: the benefits of the specialized and optimized code should amortize the compilation price.

8. Conclusion

This work presented a powerful DSL language aimed at metaprogramming and showed its application to the domain of heterogeneous embedded systems. Although the library misses some features (as noted in Further Work section), it constitutes a proof of concept that the idea of dynamic code generation is perspective and useful in the real-world scenarios

9. Further Work

The work can be continued in several directions.

The library does not provide facilities for loading on the targets existing compiled code, for example, libraries. To be applicable to a wider range of use cases it requires support of this functionality.

The development of the DSL is another direction. It can be extended with additional language constructs, for example, switch, goto or to support recursion. It can also be further developed to include more features of functional programming languages, e.g. functions as first-class citizens. Support for a debugging in terms of the DSL (breakpoints, tracing) can also be added.

References

[1]. K. Melentev, R. Belkov, and I. Kirilenko. The programming system for cybernetic heterogeneous architectures using LLVM. In Proc. of the Second Conference on Software Engineering and Information Management (SEIM-2017), 2017, pp. 31-35 (in Russian).

[2]. J. E. Stone, D. Gohara, and G. Shi. OpenCL: A parallel programming standard for heterogeneous computing systems. Computing in science & engineering, vol. 12, no. 3, , 2010, pp. 66–73.

[3]. M. Garland, S. Le Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, and V. Volkov. Parallel computing experiences with CUDA. IEEE micro, vol. 28, no. 4, 2008.

[4]. T. Rompf, K. J. Brown, H. Lee, A. K. Sujeeth, M. Jonnalagedda, N. Amin, G. Ofenbeck, A. Stojanov, Y. Klonatos, M. Dashti et al. Go meta! A case for generative programming

and DSLs in performance critical systems. In LIPIcs-Leibniz International Proceedings in Informatics, vol. 32, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[5]. K. J. Brown, A. K. Sujeeth, H. J. Lee, T. Rompf, H. Chafi, M. Odersky, and K. Olukotun. A heterogeneous parallel framework for domain-specific languages. In Proc. of the 2011 International Conference on Parallel Architectures and Compilation Techniques (PACT), IEEE, 2011, pp. 89–100.

[6]. A. K. Sujeeth, K. J. Brown, H. Lee, T. Rompf, H. Chafi, M. Odersky, and K. Olukotun. Delite: A compiler architecture for performance-oriented embedded domain-specific languages. ACM Transactions on Embedded Computing Systems (TECS), vol. 13, no. 4s, 2014, article no. 134.

[7]. T. Rompf and M. Odersky. Lightweight modular staging: a pragmatic approach to runtime code generation and compiled DSLs. Communications of the ACM, vol. 55, no. 6, 2012, pp. 121–130.

[8]. C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04), Mar 2004, pp. 75-86.

[9]. K. Czarnecki, J. T. O'Donnell, J. Striegnitz, and W. Taha. DSL implementation in MetaOCaml, Template Haskell, and C++. In Domain-Specific Program Generation, LNCS, vol. 3016, 2003, pp. 51-72.

Библиотека программирования гетерогенных архитектур

Г.В.Киргизов <gkirgizov@gmail.com>
Я.А.Кириленко <y.kirilenko@spbu.ru>
Кафедра системного программирования,
Санкт-Петербургский государственный университет,
199034, Россия, г. Санкт-Петербург, Университетская набережная, 7

Аннотация. Встраиваемые системы с гетерогенной архитектурой, рассматриваемые в данной работе, состоят из одного управляющего и одного или нескольких периферийных процессоров. Разработка ПО для таких систем представляет заметные сложности, требуя различные наборы инструментов для каждой составляющей гетерогенной системы. Достижение высокой эффективности также становится более сложной задачей. Кроме того, во многих сценариях встраиваемые системы требуют настройки во время исполнения, что непросто обеспечить с использованием стандартных средств. Эта работа представляет С-подобный предметно-ориентированный язык (DSL) для метапрограммирования и библиотеку, предоставляющую единый интерфейс для программирования периферийных процессоров с использованием этого языка. Это позволяет разрешить упомянутые проблемы. DSL встроен в C++ и позволяет свободно манипулировать написанными на нем выражениями и, таким образом, представляет собой реализацию идеи генеративного программирования, когда выразительная мощь высокоуровневого языка используется для многоступенчатой генерации низкоуровневого DSL кода. Вместе с другими возможностями, например, обобщенными DSL функциями, это делает данный язык гибким инструментом для динамической кодогенерации. Подход, используемый в библиотеке, — это динамическая компиляция. Код, написанный на предметно-ориентированном языке, транслируется в LLVM IR и затем компилируется в машинный код во время исполнения. Это открывает возможность динамических оптимизаций кода, например, специализации функций для определенных значений, известных только во время исполнения. Гибкая архитектура библиотеки обеспечивает простую расширяемость на любые платформы, поддерживаемые LLVM. В конце работы также приводятся апробация библиотеки на нескольких системах и демонстрация возможности динамических оптимизаций.

Ключевые слова: метапрограммирование; кодогенерация; встроенный DSL; гетерогенные системы; встроенные системы.

DOI: 10.15514/ISPRAS-2018-30(4)-3

Для цитирования: Киргизов Г.В., Кириленко Я.А. Библиотека программирования гетерогенных архитектур. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 45-62 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-3

Список литературы

[1]. К. Мелентьев, Р. Белков и Я. Кириленко. Система программирования кибернетических гетерогенных архитектур с использованием LLVM. Second Conference on Software Engineering and Information Management (SEIM-2017) (short papers), 2017, стр. 31-35.
[2]. J. E. Stone, D. Gohara, and G. Shi. OpenCL: A parallel programming standard for heterogeneous computing systems. Computing in science & engineering, vol. 12, no. 3, , 2010, pp. 66–73.
[3]. M. Garland, S. Le Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, and V. Volkov. Parallel computing experiences with CUDA. IEEE micro, vol. 28, no. 4, 2008.
[4]. T. Rompf, K. J. Brown, H. Lee, A. K. Sujeeth, M. Jonnalagedda, N. Amin, G. Ofenbeck, A. Stojanov, Y. Klonatos, M. Dashti et al. Go meta! A case for generative programming and DSLs in performance critical systems. In LIPIcs-Leibniz International Proceedings in Informatics, vol. 32, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
[5]. K. J. Brown, A. K. Sujeeth, H. J. Lee, T. Rompf, H. Chafi, M. Odersky, and K. Olukotun. A heterogeneous parallel framework for domain-specific languages. In Proc. of the 2011 International Conference on Parallel Architectures and Compilation Techniques (PACT), IEEE, 2011, pp. 89–100.
[6]. A. K. Sujeeth, K. J. Brown, H. Lee, T. Rompf, H. Chafi, M. Odersky, and K. Olukotun. Delite: A compiler architecture for performance-oriented embedded domain-specific languages. ACM Transactions on Embedded Computing Systems (TECS), vol. 13, no. 4s, 2014, article no. 134.
[7]. T. Rompf and M. Odersky. Lightweight modular staging: a pragmatic approach to runtime code generation and compiled DSLs. Communications of the ACM, vol. 55, no. 6, 2012, pp. 121–130.
[8]. C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO’04), Mar 2004, pp. 75-86.
[9]. K. Czarnecki, J. T. O’Donnell, J. Striegnitz, and W. Taha. DSL implementation in MetaOCaml, Template Haskell, and C++. In Domain-Specific Program Generation, LNCS, vol. 3016, 2003, pp. 51-72.

Criteria for software to safety-critical complex certifiable systems development

N.K. Gorelits <nkgorelits@2100.gosniias.ru>
A.S. Gukova <asgukova@2100.gosniias.ru>
E.V. Peskov <evpeskov@2100.gosniias.ru>
State Research Institute of Aviation Systems,
7, Viktorenko Str, Moscow, 125319, Russia

Abstract. Nowadays there is an actual problem in aviation industry – how to make the development of complex safety-critical systems certifiable according to international and domestic standards and regulations like DO-178C, DO-254, ARP 4754A, ARP 4761 etc. In the article configuration management process from the development lifecycle of DO-178C is considered as the main source of criteria for the development tool selection. Selected criteria can be applied to software tool, which supports entire development lifecycle of aviation software, as well as to software tools supporting some individual lifecycle processes. The activities of configuration management process provide work with all project lifecycle data, its storage, integrity, security, manageability and information support for data exchange between the remaining lifecycle processes, maintenance of the history of changes etc. Compliance with the principles of the configuration management process allows project managers to control development, ensure the required quality and reliability of the product; also, its certifiability and the necessary level of confidence in security, reduce financial and time development costs. As example of using criteria one of the most widely known in industry software tool for requirements development and management was analyzed for compliance with the chosen criteria.

Keywords: DO-178C; qualification requirements 178C; software development; software analysis; software choosing; certifiable systems; complex systems; complex systems development; avionics; on-board equipment; lifecycle processes; lifecycle; configuration management; system engineering.

DOI: 10.15514/ISPRAS-2018-30(4)-4

For citation: Gorelits N.K., Gukova A.S., Peskov E.V. Criteria for software to safety-critical complex certifiable systems development. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 4, 2018, pp. 63-78. DOI: 10.15514/ISPRAS-2018-30(4)-4

1. Introduction

This research was inspired by acquaintance and very productive work communication with untimely gone Michael Saburov. Michael Saburov participated in development of Russian analogs of certification standards and regulations DO-

178B [1], DO-178C [2], DO-254 [3] and DO-330 [4]. Also Michael Saburov participated in implementation of processes from these regulation documents in several industry enterprises. Michael Saburov took an active part in formation of the concepts of research described in this article. All results and experience gained by us during work on this research are dedicated to Michael Saburov.

Development and the following certification of complex safety-critical systems in compliance recommendations of regulation documents DO-178C, DO-254, ARP 4754A [5], ARP 4761 [6] is an actual task and a big challenge for modern Russian aviation industry.

Today among the software announced by its developers as supporting lifecycle of complex systems development a huge number of products are presented to allow software development in accordance with international quality standards.

Nevertheless at the moment assessment of the capabilities of each tool (or often it will be the whole product line of expensive tools) and making a reasonable choice is rather difficult problem.

Big quantity of existing software tools and systems positioned by developers as tools, which support lifecycle processes of complex systems development, don't have well-founded assessments.

Assessments and reviews about such software, based on experience of practical usage in industry projects, are very important – software market proposes a lot of software tools and systems made by Russian and foreign developers. So that's why industrial enterprises have to make difficult choice of software tools for development and the following certification of their critical-safety systems.

It is difficult to choose instrumental environment for support the entire development lifecycle – unfortunately universal multipurpose tool, which would satisfy the requirements of all standards of all industries, does not exist yet.

In general, most of the enterprises use separate tool for support and automate each process of development lifecycle (like requirements development process, configuration management processes, verification etc.). The situation is complicated because often all or the most parts of such software suite have different manufacturers. If the project have big set of weakly integrated software, then product development becomes more and more complex both in atomic tasks of individual specialist and in global meaning of the whole project – labor intensity increases.

The organization of development landscape as a bunch of software tools entails difficulties with tools integration, training costs, implementation costs, purchase of licenses. All these changes increase the amount of resources, which are needed for successful completion of the processes – human resources, financial, and time resources. In this case, reaching project goals, formulated before the beginning of work, become more and more difficult task.

In conditions of State program of import substitution [7] software tools and systems made by Russian developers cause big interests. However, usage experience of

Russian software and consequently number and reliability of assessments according to requirements listed above standards and regulations are not big enough.

In this article, we tried to understand and present what mechanisms and features software tool should have to be useful to simplify and systematize development of certifiable aviation software. This article is a part of series of materials about aviation standards research in context of choosing software tools for certifiable aviation software development [8].

2. DO-178C processes and the role of configuration management process among them

Russian analogue of DO-178C - Qualification requirements 178C [9] – regulates processes of certifiable development of aviation software. The heading of Russian document contains important words – “Requirements to the software of on-board equipment and systems at certification of aircraft”. These words uniquely determine goals of recommendations, specified in the document.

Certifiability of product – significant property, because the purpose of most developments is the following release of end product the on relevant market. In the context of aviation systems certifiability means that aircraft with system included will receive type certificate [10].

Under certifiability assurance, we mean the implementation of the development processes in specific way –

- all necessary for certification activities are performed,
- all necessary for certification objectives are achieved,
- all necessary data is collected about development process and its result,
- this data is stored and processed in such a way that certification authority could receive any data at any stage of project in order to examine the data and to trace the history of their interactions and relationships.

Activities and objectives to airborne systems and equipment development are described in document DO-178C (Russian analogue – qualification requirements 178C). DO-178C provides instructive materials and guidance to create airborne systems and equipment. Implementation of activities and objectives achievement listed in DO-178C give a chance to get in the end the result, which **performs its intended function with a level of confidence in safety that complies with airworthiness requirements**.

DO-178C describes a set of development lifecycle processes for aviation systems and equipment. DO-178C divides processes of the development lifecycle to three groups. The first group includes only one process – **software planning process**. The second group called **software development processes** includes four processes – software requirements process, software design process, software coding process and integration process. The third group consists of four **integral processes** –

software verification process, software configuration management process, software quality assurance process, certification liaison process.

During software development, processes directly creation of software of aviation systems takes place along with all previous and accompanying it measures for the design, coding, integration etc. The main result of development processes is the executable object code and its associated additional data are produces and loaded into the target hardware for further integration. This result is necessary to be achieved having carried out all the measures described in qualification requirements.

Integral processes play a role of enabling processes (by analogy with enabling systems in the terms of System Engineering [11]) - created and edited during development processes data is stored and processes through mechanisms and activities of configuration management process, required reviews and analyses are made in the verification process and so on. Data – development lifecycle artifacts or configuration items – may be requirements with different levels of details, software architecture, source code and executable object code and different protocols, problem reports, and many other results of activities.

Explanation the importance of integral processes implementation is very simple – otherwise it is very difficult almost impossible to collect necessary for certification data and to control the development process. It means that it will be difficult to provide **necessary level of confidence in safety that complies with airworthiness requirements**.

Each of integral processes has its own role and importance in the development lifecycle; it could not be ignored or partially abolished during lifecycle. Huge risks await developers who dare not comply integral processes - certification authority will not accept results obtained this way and will not give relying certificate. Also final product may contain errors and defects of varying degrees of critically. This situation will not allow achieving the required level of confidence in safety and quality of result in total, if the development process comes to an end with the release of the working result.

In modern world of computers and upcoming information technologies the whole software development lifecycle (and aviation software is not an exception) passes through software tools, information systems and therefore its databases and repositories. These software tools and information systems for all kinds of operations on data (creation, storage, editing etc.) must be evaluated for their sustainability and compliance with development according to certain standards and other regulation documents.

If perform analyze requirements to development product, which Qualification requirements 178C specifies and requires developer, becomes obvious that the most restrictions and requirements for software (in which aviation software will be developed) come from configuration management process. Activities of configuration management process provide operations with development lifecycle data, its storage, informational support to data exchange between other lifecycle processes, logging the history of changes etc.

In this research, we chose configuration management process as the source of arguments or justifications for choosing of software tools on which certifiable aviation software will be developed. These substantiations are formulated in the form of criteria. Criteria can be applied to potentially interesting software tools and systems from the market and help with assessment and reasonable choice of some of them. There will be described below how to apply selected criteria to the most widely used (worldwide and also in Russia) software for requirements development in the industry.

3. Basic criteria to tool from configuration management process

Configuration management process in project must be performed in accordance with the document "Software Configuration Management Plan". Software Configuration Management Plan should be developed for each software development project during Software Planning Process if development corresponds to Qualification requirements 178C. In this document configuration management environment should be determined as well as configuration management process activities which will be performed during software development lifecycle.

Configuration management environment must support activities from section 7.2 of Qualification requirements 178C. The list of configuration management activities contains some process regulations (which restrain project members within the workflow) and requirements to the mechanisms of configuration management environment. It would be very useful if such mechanisms and methods will be implemented in software, which will be used for development because not all of them could be replaced with some organizational regulations.

Configuration management plan contains some requirements to configuration management activities follow-up. As examples of these requirements can be listed: states of configuration items, workflows of problem reports and change requests, inspection procedures, baseline definition rules and rules of versioning configuration items, organizational restrictions, safety details etc. These requirements will not be considered in this article because its implementation can be realized regardless of the instrumental part of configuration management environment.

In this article, we identified the basic principles and mechanisms (basic criteria) determined by configuration management environment and configuration management activities according Qualification requirements 178C.

First of all we would like to highlight single and unified storage for all lifecycle data as basic configuration management principle. It means that project should have unified configuration management system for registration, storage and delivery all software development lifecycle data.

Let us enumerate basic mechanisms of configuration management environment:

- identification of configuration,
- configuration status accounting,
- change management and control,
- traceability,
- versioning,
- registration of inconsistencies and corrective actions,
- storage, retrieval and release.

Described mechanisms (further, criteria) are based on text Qualification requirements 178C and are advisory in nature. These criteria can be used as an additional informational source while choosing software tool for certifiable aviation software development.

Elements of the criteria list will be considered in more detail below.

3.1 Identification of configuration and its configuration items

Procedure of identification of the configuration item (and the whole configuration in general) includes assigning an identifier to the configuration item and registering it in the configuration management system. The identifier of configuration item is a designation uniquely distinguishes one configuration item from another. Identifier of configuration item could not be changed ever. Identifier of configuration together with its version makes unique identifier of configuration item in a particular configuration. Version of configuration item will be described below in one of the criteria.

An example of attributes that we suppose useful for registration of configuration item:

- configuration item identifier (doesn’t change ever after registration),
- mnemonics (designation which will help user identify configuration item),
- configuration item name,
- purpose of configuration item (type),
- kind of configuration item (atomic, composite – configuration index),
- version (number, sign if it is baseline or not),
- data control category (Control Category 1 or Control Category 2),
- link to the configuration item source.

Note: software lifecycle data can be classified to Data Control Category 1 or to Data Control Category 2 (section 7.3 of Qualification requirements 178C).

3.2 Configuration status accounting

Status accounting of developing software configuration should be conducted in order to provide the certification authority all necessary information (like configuration index, history of configuration etc.). That is why it is necessary to ensure that registration of the actions performed on the configuration units is automatic.

An example of data which we suppose to necessarily register when performing any action on the configuration item:

- date and time of making changes to the configuration item,
- number of version of the configuration item,
- user id – who made changes to configuration item or created version of configuration item,
- status of version of configuration item including the history of this status changes,
- for configuration items from Control Category 1: link to the change request for this configuration item.

3.3 Versioning, baselines

Rules of naming and versioning for configuration items should be defined.

Note: for example, configuration item's version is denoted as an integer (1, 2, 3 etc.). New value of configuration item's version is obtained by increasing the value by 1. If it was 2, the next value will be 3.

Rules for baseline formation and baseline appointment mechanism should be defined. In addition, restrictions on the baseline's modification should be defined.

Note: baseline is approved and registered version of configuration item, which will be used as basic for further development. Baseline can consist of one or several configuration items.

3.4 Configuration items traceability

Traceability requirements and mechanisms should be defined for link different types of configuration items and related data. Configuration items can be connected with each other, also with reason of creation (source), with dependent items, with history of configuration item's changes etc.

Note: As example of connections, we may mention links between low level requirements with its parent high level requirements, low level requirements with executable object code, problem report with configuration item, problem report with change request and with task for making approved changes etc.

Configuration items traceability is very important in the context of developing software certification. It is necessary for configuration items to trace links with source of its creation with maked to configuration items changes and with reason to making changes etc.

Traceability of links should work in both directions. Changes in configuration items should trace to sources of changes (for example to change request, which in its turn refers to parent problem report) and back.

It is always useful for users to analyze some visualized view of data. As a variant of useful and intuitive view of links and traces may be a traceability matrix. Traceability matrix shows how configuration items are connected to each other and their relations type is displayed. Type of relations between configuration items can be presented both in simple form with only displaying link presence or absence, and in the various types of links and communication.

Table 1 illustrates an example of configuration item's baseline formation.

Table 1. An example of traceability matrix: links between configuration items

Configuration items	***CI1***	***CI2***	***CI3***	***CI4***	**…**
CI1	■	X	⇅		
CI2	X	■		⬇	
CI3	⇅		■		
CI4		➡		■	
…					■

■ – not applicable
X – connection exists
⬇, ⇅, ➡ – certain type of connection exists

3.5 Change management and control

The change management of the configuration items must be implemented. Change management activities are responsible for the reaction to recording, evaluating, solving problems through the whole lifecycle of each configuration item.

Any change of configuration item should only be done by creating a new version of changing configuration item. However all previous versions should remain unchanged. Previous versions should be stored in repository and be accessible.

Changing of configuration items from Control category 1 is possible only through special procedure of change management. Problem report should be created and approved, detailed change requested and tasks should be created from this problem report. Changes to configuration items from change request should be also approved and only then changes may be applied to configuration items. All related information about changes must be stored forever – who, when, for what reason have changed that version of configuration item. Changes to configuration items with Control category 2 do not require complex procedure with approvals and reviews of changes.

3.6 Registration of inconsistencies and corrective actions

Once inconsistencies or defects are detected, it is necessary to determine procedure and mechanisms of its registration. Also corrective actions should be established, impact analysis of the proposed changes should be done and making of the approved changes to configuration item should be strictly controlled.

Any project member who discovered an inconsistency or defect or any other type of error, should be able to write it in special configuration item – problem report.

An example of attributes, which we suppose to necessarily register when registering a problem report for any configuration item:

- link to configuration item – source of detected inconsistencies,
- link to index of configuration which includes configuration item with inconsistence or to process or workflow if inconsistence is more global,
- inconsistence description,
- problem report's author id,
- steps to reproduce the problem,
- problem report state,
- link to corrective actions (for example: change request).

An example of attributes which we suppose to necessarily register when registering a corrective action for any problem report (for example: change request):

- link to problem report (change request source),
- link to configuration items in which it is necessary to make changes,
- impact analysis of proposed changes to the rest configuration items of lifecycle data.

3.7 Storage, retrieval and release

Method and proof of data integrity should be determined during its storage and retrieval from backups. Rights to release data should also be defined. Tools for creation, retrieval and integrity control of backups should be implemented according to chosen method.

Note: the need for backup creation can be both for the entire repository and for a separate development project or for separate configuration.

The realization of instrumental support for the creation, retrieval and data integrity control is very important and in demand because it allows to minimize time costs for these procedures and to reduce the risk of data distortion or loss.

Note: using of a checksum mechanisms for backups creation may be a good example of data integrity control realization.

4. Configuration management tools, analysis

The experience of cooperation with Russian developers of avionics system demonstrates that most of them try to create on-board software in compliance with the requirements of the document Qualification requirements 178B/C and then certify their software products.

At the same time there are situations when the software development process is produced without detailed requirements (in fact without requirements at all - only high-level technical specification are used), without configuration management, without reviews or inspections. Software testing is conducted, but unfortunately, its completeness can be insufficient because of the absence or incompleteness of requirements.

Realizing their unpreparedness for further certification without using of specialized software, aviation enterprises are implementing various tools. An example of such tools can be IBM Rational DOORS, IBM Rational Change + Synergy, IBM Rational Team Concert, Siemens Team Center Requirements, LDRA and others. In this case often overlooked that without understanding the processes (and not having the described processes on a paper at least) it is almost impossible to get the effect of the implementation of the tool.

It is necessary to apply the certification process with a complex approach to achieve the best result. It means - to develop the processes, to provide their support by tools, to develop plans and standards (Plan for Software Aspects of Certification, Software Development Plan, Software Verification Plan, Software Configuration Management Plan, Software Quality Assurance Plan; Software Design Standards, Software Code Standards, Software Requirements Standards) and to conduct development in full compliance with these plans and standards.

Often enterprise of the aviation industry implement only tool for writing and storage requirements. Typically, this tool has minimal change management capabilities. Developers try to manage requirements ignoring or paying low attention to the configuration management process – this approach is fundamentally incorrect.

Below we put a list of the most widely used tools to support the software development lifecycle, implemented in Russian aviation enterprises.

To support requirements management processes are often used: Microsoft Excel / Word, IBM Rational DOORS, Siemens TeamCenter Requirements Management (mainly in those enterprises where Siemens TeamCenter PLM was previously implemented in the design department) and even more rare - 3SL Cradle.

Due to the State program of import substitution, products of Russian developers arouse great interest. Among the most ambitious, it is possible to highlight product, which supports the entire development lifecycle of systems - Devprom.

To support lifecycle data change management processes are often used: IBM Rational Change + Synergy (tools are not supported by the vendor, but are still in use in some enterprises), IBM Rational Team Concert, and the most popular project and task management tools - Redmine and Attlassian Jira.

In situation when the software product Redmine or Jira are used to manage changes to the lifecycle data, the integration between these tools is rather nominal – all tools supported development lifecycle work independently, links between change requests and requirements are fixed in a text file.

This approach does not contradict the principles of configuration management prescribed in Qualification requirements 178C, but not only doesn't simplify the development process, but also makes the process management even more difficult (dependence on the human factor, the inability to track changes (the absence of a change marker), the lack of quick switch from a change request to the changed data, etc.).

To support configuration management processes are often used: GitHub - the most popular and freely distributed tool among code developers and SVN (Subversion)– a traditionally used repository for file sharing in enterprises in Russia (also distributed under the conditionally free Apache license).

The functionality of these tools when it used as configuration management systems does not allow you to fully support all activities of the configuration management section 7.2 of Qualification requirements 178C. Moreover, the use of all the functionality of this software may be considered as a violation of some of them. It is almost impossible to restrict the functionality of tools that are useful to traditional code developers in order to comply with the process specified in the Configuration Management Plan.

For example, GitHub does not store intermediate versions when you merge code branches (or other files when you use this tool as a configuration management environment) and you cannot track changes that precede the merge.

Quote from DO-178C (section 7.2.4 e): "Throughout the change activity, software life cycle data affected by the change should be updated and **records should be maintained** for the change control activity".

For the analysis for compliance with the criteria described in the previous section, we present the summarized results of the requirements management tool IBM Rational DOORS use in State Research Institute of Aviation Systems (GosNIIAS) and the results of the analysis of the entire IBM Rational product line for lifecycle management [12].

We can analyze requirements management tools for conformity by Configuration Management process criteria, because the requirement is one type of configuration items and recommendation of section 7.2 of Qualification requirements 178C about its storage and handling must be observed.

To evaluate the criteria, the following values (weight) were selected:

- 0 – criteria is not supported;
- 0.5 – criteria is partially supported;
- 0.75 – criteria is supported through tool configuration, adaptation or any integration;

- 1 – criteria is fully supported.

The analysis results are shown in the figure below on Fig.1.

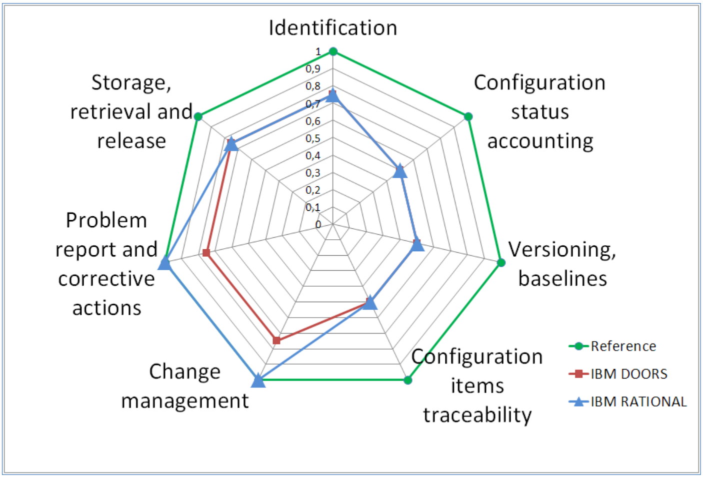


Fig. 1. Tools analysis

5. Conclusion

Configuration management process – is the main source of criteria for choosing the tools, which support aviation software development lifecycle. Configuration management process acts as unifying "input-output bus" for all lifecycle data. Therefore, tools with support of the software development lifecycle should focus on the mechanisms, embedded in the configuration management process, in order to be able to interact closely (be integrated). Such a close relationship (integration) through the configuration management process can significantly help with the development process, provide a predictable (and positive, if the tool was chosen correctly) result of aviation software development and help with preparing to the certification. It is important to note, that the purchase of the software tools and instruments does not ensure success in passing the certification – methodological support is also needed.

The task to select software tools for development lifecycle support is not easy, because it is rather difficult to determine in advance whether all requirements of chosen for this project lifecycle process will be supported by software tool, system or a set of tools. Analysis of configuration management process and selecting criteria from it to tools allows to define the boundaries of necessary for the project systems and tools. Analysis gives as result formulated requirements to the tool, which can be applied for choosing and buying suitable tool or in case of independent development such instrumental environment. In case of buying these requirements and criteria will help to choose exactly that product whose functions are necessary and sufficient for development goals without spending a lot of money for buying disparate software tools of different manufacturers, which will complicate the solution as a whole.

These conclusions are confirmed by the above analysis of one of the tools. Using of the set of tools extending the functional brings the environment closer to the reference state of configuration management process. In addition, there are difficulties: often the cost of licensing significantly increases (you have to buy additional tools), the time for installation, integration and implementation of the process increases, number of tools used in the project is growing and requires management efforts. As a result, the total complexity of development increases.

References

[1]. Software Considerations in Airborne Systems and Equipment Certification (RTCA DO-178B), 1992.

[2]. Software Considerations in Airborne Systems and Equipment Certification (RTCA DO-178C), 2011.

[3]. Design Assurance Guidance for Airborne Electronic Hardware (RTCA DO-254), 2000.

[4]. Software Tool Qualification Considerations (RTCA DO-330), 2011.

[5]. Aerospace recommended practice. Guidelines for development civil aircraft and systems (SAE ARP 4754A), 2010

[6]. Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment (SAE ARP 4761), 1996

[7]. The Order of the Ministry of Industry and Trade of the Russian Federation of March 31, 2015 № 663 “About the approval of the industry plan of actions for import substitution in branch of civil aircraft industry of the Russian Federation” (with changes and additions)

[8]. Gorelits N.K., Peskov E.V., Requirements management as efficiency measure for software development in aviation industry. Sbornik trudov VIII Mezhdunarodnoy konferentsii “IT-STANDART 2017” [Proc. of VIII International conference “IT-Standard 2017”], Moscow, 2017, pp.105-113, ISBN 978-5-98597-346-4 (in Russian)

[9]. Qualification requirements part 178C, IAC, 2014 (in Russian)

[10]. M.A.Saburov, Yu.A.Solodelov, N.K.Gorelits. Development of the certifiable avionics software by the example of JetOS real time operation system. Navigatsiya, navedenie i upravlenie letatel’nymi apparatami: tezisy dokladov Tret’ey Vserossiyskoy nauchno-tekhnicheskoy konferencii [Proceedings of Third All-Rus. Scient.-Techical Konf. “Navigation, guidance and control aircraft”], Moscow, 2017, pp.241-243, ISBN: 978-5-93728-133-3 (in Russian)

[11]. System engineering — System life cycle processes (ISO/IEC/IEEE 15288:2015), 2015

[12]. Koverninsky I.V., Kan A.V., Volkov V.B., Popov Yu.S., Gorelits N.K. Practical experience of software and system engineering approaches in requirements management for software development in aviation industry. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 2, 2016, pp.173-179. DOI: 10.15514/ISPRAS-2016-28(2)-11

Критерии, предъявляемые к программному обеспечению для разработки сложных сертифицируемых систем, критичных по безопасности

Н.К. Горелиц < nkgorelits@2100.gosniias.ru>
А.С. Гукова < asgukova@2100.gosniias.ru>
Е.В. Песков <evpeskov@2100.gosniias.ru>
Государственный научно-исследовательский институт авиационных систем
Россия, 125319, г. Москва, ул. Викторенко, 7

Аннотация. На сегодняшний день в авиационной отрасли существует актуальная проблема – как инструментально поддержать и обеспечить сертифицируемость разработки критичных по безопасности сложных систем в соответствии с международными и отечественными отраслевыми стандартами, нормативными документами, такими как КТ-178С, КТ-254, Р-4754, Р 4761 и др. В статье рассматривается процесс управления конфигурацией при разработке по КТ-178С как основной источник критериев для осуществления выбора инструментального средства поддержки разработки. Выделенные критерии могут быть применены к инструменту поддержки всего жизненного цикла разработки авиационного ПО в соответствии с КТ-178С, а также к инструментам, отвечающим за поддержку отдельных процессов жизненного цикла. Мероприятия процесса управления конфигурацией обеспечивают работу с данными жизненного цикла, их хранение, целостность, безопасность, управляемость, информационную поддержку обмена данными между остальными процессами жизненного цикла, ведение истории изменений и т.п. Соблюдение принципов процесса управления конфигурацией позволяет осуществлять контроль разработки, обеспечить требуемые качество и надежность продукта, его сертифицируемость и необходимый уровень доверия к безопасности, снизить финансовые и временные затраты на разработку. В качестве примера использования критериев приведен анализ одного из распространенных в отрасли инструментов разработки и управления требованиями на соответствие указанным критериям.

Ключевые слова: КТ-178С; DO-178C; разработка ПО; анализ ПО; выбор ПО; сертифицируемые системы; сложные системы; разработка сложных систем; авионика; КБО; процессы ЖЦ; жизненный цикл; управление конфигурацией; системная инженерия.

DOI: 10.15514/ISPRAS-2018-30(4)-4

Для цитирования: Горелиц Н.К., Гукова А.С., Песков Е.В. Критерии, предъявляемые к программному обеспечению для разработки сложных сертифицируемых систем, критичных по безопасности. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 63-78 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(4)-4

Список литературы

[1]. Квалификационные требования часть 178В, 2002 – АР МАК.

[2]. Квалификационные требования часть 178С, 2014 – АР МАК.
[3]. Руководство по гарантии конструирования бортовой электронной аппаратуры КТ-254, 2011 – АР МАК.
[4]. Software Tool Qualification Considerations (RTCA DO-330), 2011.
[5]. Руководство Р4754 по процессам сертификации высокоинтегрированных сложных бортовых систем воздушных судов гражданской авиации. АР МАК, 2010
[6]. Руководство 4761 по методам оценки безопасности систем и бортового оборудования воздушных судов гражданской авиации, 2010
[7]. Приказ Министерства промышленности и торговли РФ от 31 марта 2015 г. N 663 "Об утверждении отраслевого плана мероприятий по импортозамещению в отрасли гражданского авиастроения Российской Федерации" (с изменениями и дополнениями)
[8]. Горелиц Н.К., Песков Е.В., "Управление требованиями как критерий эффективности при разработке программного обеспечения в авиационной отрасли", Сборник трудов VIII Международной конференции "ИТ-Стандарт 2017". Москва, 2017, стр. 105-113, ISBN 978-5-98597-346-4
[9]. Software Considerations in Airborne Systems and Equipment Certification (RTCA DO-178C), 2011
[10]. Сабуров М.А., Солоделов Ю.А., Горелиц Н.К. Разработка сертифицируемого бортового программного обеспечения на примере операционной системы реального времени JetOS. Навигация, наведение и управление летательными аппаратами: тезисы докл. Третьей Всерос. научно-технической конф. (Москва – Раменское 21-22 сент. 2017 г.), 2017, стр. 241-243, ISBN: 978-5-93728-133-3.
[11]. ГОСТ Р 57193 Системная и программная инженерия. Процессы жизненного цикла систем, 2016
[12]. Koverninsky I.V., Kan A.V., Volkov V.B., Popov Yu.S., Gorelits N.K. Practical experience of software and system engineering approaches in requirements management for software development in aviation industry. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 2, 2016, pp.173-179. DOI: 10.15514/ISPRAS-2016-28(2)-11

Towards formal verification of cyber security standards

Tomas Kulik <tomaskulik@eng.au.dk>
Peter Gorm Larsen <pgl@eng.au.dk>
Aarhus University, Department of Engineering
Finlandsgade 22, Aarhus, 8200, Denmark

Abstract. Cyber security standards are often used to ensure the security of industrial control systems. Nowadays, these systems are becoming more decentralized, making them more vulnerable to cyber attacks. One of the challenges of implementing cyber security standards for industrial control systems is the inability to verify early that they are compliant with the relevant standards. Cyber security standard compliance is also only validated and not formally verified, often not providing strong proofs of correct use of cyber security standard. In this paper, we propose an approach that uses formal analysis to achieve this. We formally define building blocks necessary to define the system formally in order to enable formal modeling of the system and carry out the analysis using the Alloy Analyzer. Our approach can be used at an early design stage, where problems are less expensive to correct, to ensure that the system has the desired security properties. We show the applicability of our approach by modeling two distinct cyber attacks and mitigations strategies used to defend against these attacks and also evaluate our approach based on its flexibility to handle and combine different aspects of the cyber security standards. We discuss the future directions of our research.

Keywords: cyber security; formal analysis; cyber security standards.

DOI: 10.15514/ISPRAS-2018-30(4)-5

For citation: Kulik T., Larsen P.G. Towards Formal Verification of Cyber Security Standards. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 4, 2018, pp. 79-94. DOI: 10.15514/ISPRAS-2018-30(4)-5

1. Introduction

In an industrial setting there is an increasing use of wireless technology because many components becomes Internet of Things (IoT) enabled. Rather than having to investing in a continuation of wired connections the balance between cost and agility many companies moves to such IoT solutions. However, this move towards wireless technologies gives new security challenges that must be taken serious in order to protect both the data and algorithms owned by the companies. In order to ensure this different security standards have emerged and here the IAS/IEC-62443 is a promising candidate that deserves special examination [1].

In order to master the increase of complexity caused by the increased wireless connections the architectures of the distributed systems needs thorough analysis. Here model checking is a promising candidate to provide such an analysis. This has an appropriate balance between the time and cost spent on the analysis and the exhaustiveness favorable. In this paper we demonstrate how this can be achieved defining possible attacks and the corresponding mitigations using a formal approach. The main result is an illustration of how this kind of framework can be deployed to illustrate how a specific architecture and its chosen mitigations can be proven that the different cyber-attacks cannot be realized.

The remainder of this paper is structured as follows: Section 2 introduces the essential parts of the ISA/IEC-62443 standard and this is followed in Section 3 defining the architecture of considered system. The main result of this paper is presented in Section 4 defining extended formal framework for cyber-attacks and possible mitigations for these. Section 5 explains about how formal analysis can be conducted using the Alloy Analyzer [2]. This is followed by Section 6, which considers related work for formal analysis of cyber security standards. Finally, Section 7 provide concluding remarks also indicating the future directions planned for this work.

2. The chosen cyber security standard

Within this paper we consider security of an industrial control systems based on IoT environment. This is further considered in terms of applying cyber security standards that are used to ensure industrial automation and control system security, specifically the ISA/IEC-62443 series of standards.

The series is split into 4 distinct groups where each group considers different perspective of cyber security of the industrial automation control system (IACS). Each of the groups contain documents, where each document is understood as a single standard. This leads to name designation of specific standards based on the format: ISA/IEC-62443-*X*-*Y* where *X* is the designation of the group and *Y* is the designation of the specific document.

The first group, **ISA/IEC-62443-1, General**, considers the general aspects of the standard and cyber security. Concepts and metrics defined within this group are present throughout the other groups of the standard as shown in Fig. 1. The second group, **ISA/IEC-62443-2, Policies and procedures**, focuses on organizational aspects of cyber security. The main consideration of this group is providing the requirements that the organization has to fulfill in order to manage their cyber security program. The third group, **ISA/IEC-62443-3, System**, addresses the security on a system level. The security requirements for the system is defined here as well as guidance on implementation of these and fulfillment of these requirements. The final group, **ISA/IEC-62443-4, Component**, contains documents defining detailed requirements for cyber security on the component level.

2.1 The standard under consideration

The standard that we consider for formal verification is ISA/IEC-62443-3-3, System security requirements and security levels. This standard has been selected as it provides requirements that are applicable on system level and are verifiable by technical means. The intended audience for this standard are asset owners, system integrators and service suppliers and the purpose of this standard is to use the defined requirements to evaluate the system under consideration and determine if this system is capable of reaching a specific security level (SL). The standard defines 4 SLs:

- **SL 1**: The lowest SL aimed to prevent unauthorized disclosure of information via eavesdropping or casual exposure.
- **SL 2**: Aimed to prevent unauthorized disclosure of information to an entity actively searching for it using simple means with low resources, generic skills and low motivation.
- **SL 3**: Aimed to prevent unauthorized disclosure of information to an entity actively searching for it using sophisticated means with moderate resources, IACS specific skills and moderate motivation.
- **SL 4**: The highest SL aimed to prevent unauthorized disclosure of information to an entity actively searching for it using sophisticated means with extended resources, IACS specific skills and high motivation.

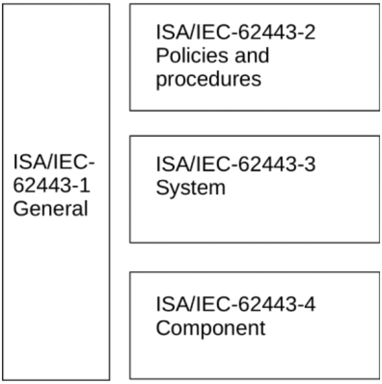


Fig. 1. Overview of ISA/IEC-62443 series structure

Within the standard the security requirements on the system level are considered as system requirements (SRs) where each SR can define 0 to 3 requirement enhancements (REs). SL of the aspect of the system is measured as a compliance with SRs and REs for this aspect, shown in Table 1.

Table. 1. Mapping between compliance with SRs, Res and corresponding SLs

SR	RE(s)	SL
SR 1	None	SL 1
SR 1	RE 1	SL 2
SR 1	RE 1 + RE 2	SL 3
SR 1	RE 1 + RE 2 + RE 3	SL 4

In case that no SR is defined for the given aspect of the system, the standard implicitly defines SL 0 as an SL for this aspect of the system.

3. System architecture

The system under consideration extends a generic control systems architecture and capabilities defined in the framework for Threat-driven Cyber Security Verification of IoT Systems (FCSVIoT) [3]. This architecture consists of subsystems equipped with sensors and actuators shown on Fig. 2. Each subsystem is a microcontroller capable of computation and communication. Communication between the subsystems creates a distributed control system, which provides data to and accepts commands from a central engineering terminal. In this paper we extend the architecture with the notion of **router**, a special type of subsystem that enables data exchange among other subsystems and extends the capabilities of the system by defining user actions on the engineering terminal. We further consider that communication channels must exist between subsystems in order to exchange data.

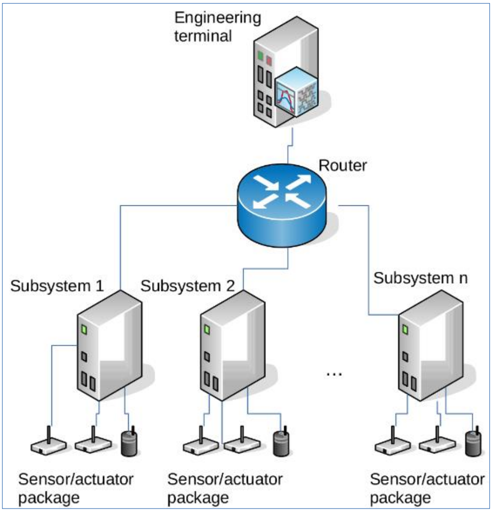


Fig. 2. Architecture of the system under consideration

We let our subsystems be governed by a set of atomic actions forming a basic alphabet for each subsystem S_i as $SA = \{generate, send, acquire, accept, discard, connect, disconnect, recover, compromise\}$ and each subsystem has a finite set of states S. Actions cause transitions between states of the subsystem such as:

$$s \xrightarrow{action(param)} s' where\ s, s' \in \mathrm{S}$$

We further define a predicate on communication channels *secure(c)* stating that the communication channel is secured. The generate action represents generation of data by the subsystem, send action represents sending the data on a communication channel, acquire represents acquiring data from the communication channel, accept defines accepting the acquired data, discard defines discarding the acquired data. The connect and disconnect action represent a subsystem connecting to and disconnecting from a communication channel. The compromise action moves the subsystem to a compromised mode of operation, *compromised*(S_i), where we consider that the subsystem has malicious intent. Recover action moves the subsystem from compromised to normal mode of operation, *normal*(S_i).

We extend the actions in the FCSVIoT by considering the engineering terminal E as an user interaction part of the system by defining its own alphabet of actions $EA = \{allow, forbid\}$, where *allow* represents allowing and *forbid* represents disallowing interaction with an user by the engineering terminal. We also consider that the system holds a set of user accounts allowing users access to the Engineering terminal, *Ac* where a single account is denoted as *a*. Each account has exactly one credential *cr*, hence the system also holds a set of valid credentials *Cr*. We further define the router R as with alphabet $RA = SA \backslash \{generate\}$ as the router is not equipped with sensors to generate its own data. This leads to creation of system alphabet $A = SA \cup EA$.

4. Attacks and mitigations

We define cyber attacks as sequences of events leading to potential harm to the system under attack. Within this paper we consider two cyber attacks, specifically data packet tampering and brute force attack against an user account [4]. These attacks have been purposefully selected as the selected cyber security standard addresses them and specifies requirements for mitigations aimed to increase cost of these attacks. We provide a formal description of the attack sequence and mitigation for both of the attacks under consideration.

4.1 Data packet tampering attack

Packet tampering is the act of a compromised subsystem, specifically a router changing values in a data packet, causing the intended receiving subsystem to receive different values from those sent by the transmitting subsystem. This has then the potential to cause unsafe behavior of the system. In order to describe an instance of this attack, consider two subsystems S_0 and S_1 operating in normal mode, which we show formally as *normal*(S_0) $\wedge$ *normal*(S_1) and a router R used to enable data exchange between the two subsystems. The router operates in a compromised mode *compromised(R)*, meaning that some malicious actor has access to and control over this router.

The subsystems S_0 and S_1 are always connected to the router, meaning that at any time they can exchange data with the router using communication channels c_0 and c_1. We use the FCSVIoT predicate *always_connected* as *always_connected*(S_0, R_0, c_0) ∧ *always_connected*(S_1, R_0, c_1) specifying that there is always possibility of communication between S_0 and S_1 via R_0.

Now we consider that S_0 is sending a unit of data *d* to S_1. The data *d* is first obtained by R_0, which modifies the data *d*, represented by a new *modify* action added to the alphabet of the router in order to represent software installed by malicious actor, and then sends it further to S_1. The attack hence combines actions into a pattern by following a specific sequence:

1. S_0.*generate(d)*
2. S_0.*send*(c_0, *d*)
3. R_0.*acquire*(c_0, *d*)
4. R_0.*modify(d)*
5. R_0.*send*(c_1, *d*)
6. S_1.*acquire*(c_1, *d*)
7. S_1.*accept(d)*

Main act of the attack happens at the R_0.*modify(d)* event. Here the data *d* becomes malicious as *malicious(d)*. In case of non-existent mitigations within the system, the subsystem S_1 simply accepts the data and becomes itself compromised, hence the attack is successful.

In order to mitigate this attack, we consider security requirements from the ISA/IEC-62443-3-3 security standard, covering the communication integrity, namely SR 3.1 stating that *The control system shall provide the capability to protect the integrity of transmitted information*.

The requirement itself does not provide the necessary guidance on what method to use to protect the data, hence we consider the SR 3.1 RE 1 specifying the cryptographic integrity protection as *The control system shall provide the capability to employ cryptographic mechanisms to recognize changes to information during communication*. To mitigate the attack from a general perspective we consider that the data has to contain a cryptographic signature derived from the data content and a secret known to subsystems, but not routers. This introduces a concept of signed data, which we do by extending the alphabet of the subsystem by adding an atomic *sign* event as *sign(d)*. We further define a predicate for signed data stating that the data is considered signed only if signed by a subsystem operating in a normal mode:

$$signed(d) = \exists s: s \xrightarrow{sign(d)}{}_{i} s' \wedge s \in S_N$$

We then consider applying the signing event as a mitigation by specifying that the subsystem discards the signed data if it has been modified, with indices added to the state notation, describing the order of state transitions:

$$\forall s_1 : s_1 \xrightarrow{discard(d)}{}_i\ s_2 \text{ if}$$
$$always\_connected(S_i, R_j, c_k)$$
$$\text{and } \exists s_0 : s_0 \xrightarrow{acquire(d,c_k)}{}_i\ s_1$$
$$\text{and } \exists s : s \xrightarrow{modify(d)}{}_j\ s'$$
$$\text{and } signed(d)$$

Applying this mitigation by adding S_0*.sign(d)* after the S_0*.generate(d)* event causes the final event in the chain to be S_1*.discard(d)* since R_0*.modify(d)* is present. This means that the subsystem S_1 does not enter compromised mode and the cyber attack is unsuccessful.

4.2 Brute force attack against an user account

Brute force attack against an user account uses computational power to try to guess user sign in credentials by randomly generating passwords and user names and providing them to the system for verification. The attack can be streamlined if the user name and length of the password is known, decreasing the "guess space", which in turn leads to less time required to guess the correct credentials. If the user account can be breached this gives the malicious actor control over the system in terms that the breached account allows, potentially allowing the malicious actor submission of malicious commands to the system. To formally describe the attack we consider a single engineering terminal E_0 operating in a normal mode, *normal(*E_0*)*. We further define a *check* function on an engineering terminal, responsible for raising the *allow* or *forbid* event:

$$check(cr) = \begin{cases} allow, & \text{if } cr \in Cr \\ forbid, & \text{otherwise} \end{cases}$$

We formulate the attack as a recursive *crack(cr)* function that generates new *cr* for every attempt used to find a *cr* such that *cr* ∈ *Cr*:

$$crack(check(cr)) = \begin{cases} true, & \text{if } allow \\ crack\left(check\left(new(cr)\right)\right), & \text{if } forbid \end{cases}$$

Once the function returns true the malicious actor has obtained access to the engineering terminal, causing the engineering terminal to operate in a compromised mode of operation, as *compromised(*E_0*)* and the attack is considered successful.

In order to mitigate the attack we consider the requirement SR 1.11 defined in ISA/IEC-62443-3-3, stating, *The control system shall provide the capability to enforce a limit of a configurable number of consecutive invalid access attempts by any user (human, software process or device) during a configurable time period. The control system shall provide the capability to deny access for a specified period of time or until unlocked by an administrator when this limit has been exceeded*. To enforce this we define a locked predicate acting on specific account mapped via its valid credential where mapping between account *ac* and credential *cr* is one to one and hence for simplicity we omit *cr* and consider *ac* as belonging to a specific *cr* as:

$$locked(ac) = \neg \exists s: s \xrightarrow{allow()} s': ac \in Ac$$

We then need to consider the amount of allowed invalid access attempts. In order to abstract away from details of password complexity, we present an assumption stating that the successful brute force attack against an system that allows reasonable small amount (in general we would consider this less than 10 for practical reasons) invalid access attempts is so unlikely that we consider it impossible. Using this assumption as a mitigation we can guarantee that the user account cannot be breached by brute force attack. We also abstract away from notion of time intervals as we consider that the brute force attack is happening rapidly and would always exceed the amount of tries within a specific time interval. We formally show this mitigation by first defining a global variable for *ac* holding the current attempt as *attempt(ac)* for its credential *cr*:

$$attempt(ac) = \begin{cases} attempt(ac) + 1, & \text{if } check(cr) = forbid \\ 0, & \text{otherwise} \end{cases}$$

We then use the variable in adding an attempt limit on using a credential to sign in to an user account such that the account becomes locked if the maximum amount of attempts is reached:

$$limited(cr, max\_att) = locked(ac) \text{ if } attempt \geq max\_att$$

By applying the *limited* predicate to the credentials we cause the account to become *locked* as a result of the *crack* function. Since a locked account cannot be used to gain access to the engineering terminal, the cyber attack fails and the engineering terminal continues in the normal mode of operation, $normal(E_0)$. It is important to note that in general the *max_att* has to be set in such a way that does not hinder usability of the system, while providing assurance of sufficient security. This mitigation strategy has therefore a limitation in case the *max_att* is set unreasonably large.

5. Formal analysis

In this section we shortly present the extensions made to the FCSVIoT and show how the mitigations for data packet tampering and brute force against user account attacks have been verified when considered within the architecture defined in Section 2. This is achieved by expressing the aforementioned attacks and mitigations using FCSVIoT with extensions introduced in this paper and verifying these scenarios using the Alloy Analyzer.

5.1 Short introduction to Alloy Analyzer

Alloy is a formal specification language, based on first order logic, used for expressing structural constraints in software systems. Alloy allows for modeling at different levels of abstraction, where at the highest level it provides object oriented interpretation, at second level it uses the set theory and at the lowest level atoms and relations are used. Within our model we are using the set theory, atoms and relations to model the types using the **sig** keyword. Subtyping is supported in Alloy by usage of **extends** keyword. We model relations between objects by specifying mappings between sets, for example `has:set EngTerminal one->some Account`, where `has` is the relation stating that the **one**, meaning exactly one engineering terminal has **some**, meaning at least one account associated with it. The scope of the model is specified after the **run** block, by quantifying how many atoms do we want to include in the model by using the **exactly** keyword. Properties of the Alloy model can be verified by usage of the Alloy Analyzer software tool [5], which checks properties of the model by generating counterexamples.

5.2 Overview of extensions to FCSVIoT

Among the first extensions is addition of new data types `Router` corresponding to the router, `EngTerminal` corresponding to the engineering terminal, `Account` corresponding to user account and `Credential` corresponding to credential for specific account as specified in Section 3. Using Alloy Analyzer, we define these datatypes using set definitions, represented by the **sig** keyword. We further extend the `State` definition with number of new relations. We further define the router as an extension of `Device` type and also adapt `Subsystem` to be extension of `Device`, as shown in Listing 1.

The `Device` type is used as a base type since `Router` and `Subsystem` share most of the actions. The only difference is that we consider that the router is not capable of generating data. The relations within `State` now also use `Device` in order to model relation that cover both `Router` and `Subsystem`. For example, the `compromised` relation shown in Listing 1 shows that a state can contain any number of compromised devices. Another relation recorded in each state is for example `accepted` which maps devices to data.

The different types discussed above are governed by several facts, which are understood as constraints on the model. One of these is the consideration that the data is either signed or not and this does not change as the system progresses in its state transitions. This is shown in Listing 2. In this constraint `s'` is the state following the `s` state, hence the constraint guarantees that the data remains signed in all states.

```
open util/ordering[State]
sig Data {}
sig Device {}
sig Subsystem extends Device{}
sig Router extends Device{}
sig Channel {}
sig EngTerminal {}
sig Credential{}
sig Account{}
sig State {
...
compromised: set Device,
can_authorise: set Subsystem,
malicious: set Data,
signed: set Data,
accepted: set Device -> set Data,
secure: set Channel,
attempts_exceeded: set Account,
limited: set Account,
cracked: set Account,
large: set Credential,
locked: set Account,
has:set EngTerminal one->some Account,
hasCred:set Account one->one Credential
...
}{ /* Facts belonging to State */ ... }
```

Listing 1. Extensions and changes to the modeling framework

```
fact{
...
all s:State, s’:s.next |
s.signed = s’.signed
}
```

Listing 2. Global constraint governing signed data

5.3 Verification of data tampering mitigation strategy

Here we demonstrate the mitigation strategy applied to a scenario discussed in 4.1. The simplest model to demonstrate data tampering mitigation strategy in fact only requires one subsystem and a router as it is the router that is responsible for the attack. This is shown in Listing 3. The listing shows the constraint for mitigation and the setup of the model. The complete extended FCSVIoT can be found via [6].

```
run {
...
// mitigation signed data
all s:State | all d:Data | d in s.signed
//test the condition
some malicious
...
} for
exactly 5 State
, exactly 1 Subsystem
, exactly 1 Data
, exactly 1 Channel
, exactly 1 Router
, exactly 2 Device
, exactly 0 EngTerminal
...
```

Listing 3. Verifying the data tampering mitigation strategy using Alloy

The **run** commands checks that the data is signed in five states, required to exectute the whole scenario. The result of this execution is: `No instance found`. This means that the Alloy Analyzer could not find a counter-example within the requested scope and the mitigation strategy is proven to work.

5.4 Verification of brute force attack:

We will show how the mitigation strategy for brute force attack can be modeled by considering a scenario described in Section 4.2. The **run** command for the model we use consists of one engineering terminal with one user account, with its associated credentials and omits subsystems as shown in Listing 4. The mitigation used in this scenario states that all accounts within all states of the system are always considered limited (i.e. they consider limit on the number of unsuccessful login attempts).

This scenario considers three states, creating the smallest scope necessary for its execution. Once the **run** command is executed the Alloy Analyzer returns `No instance found`, confirming that the mitigation strategy prevents the user account from being cracked.

```
run {
...
// mitigation account has limited tries
all s:State | all a:Account |
a in s.limited
//test the condition
some cracked
//start with no cracked account
no first.cracked
...
}for
exactly 3 State
, exactly 1 EngTerminal
, exactly 1 Account
, exactly 1 Credential
, exactly 0 Subsystem
...
```

Listing 4. Verifying the brute force attack mitigation strategy using Alloy

6. Related work

As cyber security is becoming very important topic in the industry, mainly in advent of digitalization and trends such as industry 4.0 [7], research is being carried out within the area of using formal methods in order to provide proofs that systems meet cyber security requirements [8][9][10]. The benefits of using model based verification are its applicability at an early stage of system development in order to help avoid exposure to attacks as well as provide mitigations for attacks that are not easily avoidable [11][12]. This approach consists of formal description of the behavior of a system and formal description of cyber-attacks and mitigations. The complete model is then formally analyzed in order to verify that the mitigation strategies prevent the cyber-attacks from causing potentially harmful behavior of the system. Sometimes specific cyber security standards are considered as criteria for these mitigations strategies [13].

In order to provide assurance that an industrial control system meets criteria specified in a cyber security standard, authors of [14] have investigated the ISA-99.01-01 standard by considering the requirements and metrics specified within the standard. While the authors have described part of the standard formally, their goal was not to conduct formal analysis to ensure the satisfiability of the security requirements by a given architecture but rather to provide recommendations to the operators of industrial control systems to not blindly trust standards but verify their security impact on the system.

The authors of [15] have proposed a formalization and verification technique for ISO/IEC-15408 standard known as Common Criteria using Z notation. In their technique they consider the natural language definitions within the standard and create formal templates based on these. The authors suggest usage of the templates against the formalized specification of the target system, which is left to the party verifying the system against the instantiated templates. The authors provide an example of this verification using the Z/EVES theorem prover. Our approach differs by providing formal building blocks for the system from the start, hence formalization of the system can be done by selecting from these building blocks.

7. Conclusions and future work

So far this research has demonstrated that the chosen approach is quite extensible where this paper has demonstrated how the models made in Alloy can be extended in a conservative manner with additional threats. It is expected that we in the future in this context is furthering the formal definitions to encompass more of aspects of the security standard and to verify these against larger variety of cyber-attacks. We further consider switching to TLA+ [16] in order to show the applicability of our framework using different formalism.

Acknowledgments

This work is partially supported by the Manufacturing Academy of Denmark (MADE) Digital project. For more information see http://www.made.dk/.

References

[1]. International Society of Automation. The 62443 Series of Standards. http://isa99.isa.org/Public/Information/The-62443-Series-Overview.pdf, accessed on 13/3/18

[2]. D. Jackson, Software Abstractions: Logic, Language, and Analysis. Heyward Street, Cambridge, MIT Press, April 2006, iSBN-10: 0-262-10114-9.

[3]. Tomas Kulik, Peter W. V. Tran-Jørgensen, Jalil Boudjadar, and Carl Schultz. A framework for threat-driven cyber security verification of iot systems. In Proc. of the First International Workshop on Verification and Validation of Internet of Things, Västerås, Sweden, april 2018, in print.

[4]. C. Bekara. Security issues and challenges for the iot-based smart grid. Procedia Computer Science, vol. 34, 2014, pp. 532–537. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050914009193

[5]. The Alloy Analyzer Modelling website, http://alloy.mit.edu/alloy/, 2018

[6]. Tomas Kulik and Peter Gorm Larsen. Extensions to formal security modeling framework. https://github.com/kuliktomas/FCSVIoT/commit/189c7962f7f0870fa5315c31a71a6b35e896e47d, 2018.

[7]. N. Jazdi. Cyber physical systems in the context of industry 4.0. In Proc. of the 2014 IEEE International Conference on Automation, Quality and Testing, Robotics, May 2014, pp. 1–4.

[8]. M. Ge and D. S. Kim. A framework for modeling and assessing security of the internet of things. In Proc. of the 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), 2015, pp. 776–781.

[9]. C. Heitmeyer, M. Archer, E. Leonard, and J. McLean. Applying Formal Methods to a Certifiably Secure Software System. Software Engineering, IEEE Transactions on Software Engineering, vol. 34, no. 1, 2008, pp. 82 –98.

[10]. A. N. Haidar and A. E. Abdallah. Formal modelling of pki based authentication. Electronic Notes in Theoretical Computer Science, vol. 235, 2009, pp. 55 – 70. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S157106610900084X

[11]. D. C. Wardell, R. F. Mills, G. L. Peterson, and M. E. Oxley. A method for revealing and addressing security vulnerabilities in cyber-physical systems by modeling malicious agent interactions with formal verification. Procedia Computer Science, vol. 95, no. Supplement C, 2016, pp. 24 – 31, Available: http://www.sciencedirect.com/science/article/pii/S1877050916324619

[12]. F. A. Teixeira, F. M. Pereira, H.-C. Wong, J. M. Nogueira, and L. B. Oliveira. Siot: Securing internet of things through distributed systems analysis. Future Generation Computer Systems, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17304235

[13]. J. Woodcock, S. Stepney, D. Cooper, J. A. Clark, and J. Jacob. The Certification of the Mondex Electronic Purse to ITSEC Level E6. Formal Aspects of Computing, vol. 20, no. 1, 2008, pp. 5–19.

[14]. D. K. Holstein and K. Stouffer. Trust but verify critical infrastructure cyber security solutions. In Proc. of the 43rd Hawaii International Conference on System Sciences, 2010, pp. 1–8.

[15]. S. Morimoto, S. Shigematsu, Y. Goto, and J. Cheng. Formal verification of security specifications with common criteria. In Proceedings of the 2007 ACM Symposium on Applied Computing, ser. SAC ’07, 2007, pp. 1506–1512. [Online]. Available: http://doi.acm.org/10.1145/1244002.1244325

[16]. L. Lamport, Specifying systems: the TLA+ language and tools for hardware and software engineers. Addison-Wesley Longman Publishing Co., Inc., 2002, Boston, MA, USA, 384 pages.

К формальной верификации стандартов кибербезопасности

Томаш Кулик <tomaskulik@eng.au.dk>
Питер Горм Ларсен <pgl@eng.au.dk>
Орхусский университет, Департамент инженерии,
Finlandsgade 22, Орхус, 8200, Дания

Аннотация. Стандарты кибербезопасности часто используются для обеспечения защищенности промышленных систем управления. В последнее время такие системы становятся все более децентрализованными, что делает их все более уязвимыми для разного рода кибератак. Одна из проблем реализации стандартов кибербезопасности в промышленных системах управления состоит в том, что невозможно своевременно проверить, соответствуют ли разрабатываемые системы этим стандартам или нет. Помимо прочего, соответствие стандарту кибербезопасности только валидируется, а не верифицируется формально, что, как правило, не дает убедительных доказательств правильного использования стандарта. В статье предлагается подход, в котором проверка защищенности промышленных систем управления осуществляется путем формального анализа. Подход состоит в следующем: определяются строительные блоки, необходимые для формального описания системы; составляется формальная модель системы; модель анализируется с помощью инструмента Alloy Analyzer. Предлагаемый подход может использоваться на ранних стадиях проектирования, где проблемы не так дороги для исправления. Чтобы показать применимость подхода, были смоделированы две кибератаки, а также стратегии противодействия им. Подход был также оценен на предмет гибкости — возможности совмещения разных аспектов стандартов кибербезопасности. В статье также обсуждаются будущие направления исследования.

Ключевые слова: кибербезопасность, формальный анализ; стандарты кибербезопасности

DOI: 10.15514/ISPRAS-2018-30(4)-5

Для цитирования: Кулик Т., Ларсен П.Г. К формальной верификации стандартов кибербезопасности. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 79-94 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(4)-5

Список литературы

[1]. International Society of Automation. The 62443 Series of Standards. http://isa99.isa.org/Public/Information/The-62443-Series-Overview.pdf, accessed on 13/3/18
[2]. D. Jackson, Software Abstractions: Logic, Language, and Analysis. Heyward Street, Cambridge, MIT Press, April 2006, iSBN-10: 0-262-10114-9.
[3]. Tomas Kulik, Peter W. V. Tran-Jørgensen, Jalil Boudjadar, and Carl Schultz. A framework for threat-driven cyber security verification of iot systems. In Proc. of the

First International Workshop on Verification and Validation of Internet of Things, Västerås, Sweden, april 2018, in print.

[4]. C. Bekara. Security issues and challenges for the iot-based smart grid. Procedia Computer Science, vol. 34, 2014, pp. 532–537. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050914009193

[5]. The Alloy Analyzer Modelling website, http://alloy.mit.edu/alloy/, 2018

[6]. Tomas Kulik and Peter Gorm Larsen. Extensions to formal security modeling framework. https://github.com/kuliktomas/FCSVIoT/commit/189c7962f7f0870fa5315c31a71a6b35e896e47d, 2018.

[7]. N. Jazdi. Cyber physical systems in the context of industry 4.0. In Proc. of the 2014 IEEE International Conference on Automation, Quality and Testing, Robotics, May 2014, pp. 1–4.

[8]. M. Ge and D. S. Kim. A framework for modeling and assessing security of the internet of things. In Proc. of the 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), 2015, pp. 776–781.

[9]. C. Heitmeyer, M. Archer, E. Leonard, and J. McLean. Applying Formal Methods to a Certifiably Secure Software System. Software Engineering, IEEE Transactions on Software Engineering, vol. 34, no. 1, 2008, pp. 82 –98.

[10]. A. N. Haidar and A. E. Abdallah. Formal modelling of pki based authentication. Electronic Notes in Theoretical Computer Science, vol. 235, 2009, pp. 55 – 70. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S157106610900084X

[11]. D. C. Wardell, R. F. Mills, G. L. Peterson, and M. E. Oxley. A method for revealing and addressing security vulnerabilities in cyber-physical systems by modeling malicious agent interactions with formal verification. Procedia Computer Science, vol. 95, no. Supplement C, 2016, pp. 24 – 31, Available: http://www.sciencedirect.com/science/article/pii/S1877050916324619

[12]. F. A. Teixeira, F. M. Pereira, H.-C. Wong, J. M. Nogueira, and L. B. Oliveira. Siot: Securing internet of things through distributed systems analysis. Future Generation Computer Systems, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17304235

[13]. J. Woodcock, S. Stepney, D. Cooper, J. A. Clark, and J. Jacob. The Certification of the Mondex Electronic Purse to ITSEC Level E6. Formal Aspects of Computing, vol. 20, no. 1, 2008, pp. 5–19.

[14]. D. K. Holstein and K. Stouffer. Trust but verify critical infrastructure cyber security solutions. In Proc. of the 43rd Hawaii International Conference on System Sciences, 2010, pp. 1–8.

[15]. S. Morimoto, S. Shigematsu, Y. Goto, and J. Cheng. Formal verification of security specifications with common criteria. In Proceedings of the 2007 ACM Symposium on Applied Computing, ser. SAC ’07, 2007, pp. 1506–1512. [Online]. Available: http://doi.acm.org/10.1145/1244002.1244325

[16]. L. Lamport, Specifying systems: the TLA+ language and tools for hardware and software engineers. Addison-Wesley Longman Publishing Co., Inc., 2002, Boston, MA, USA, 384 pages.

Combining ACSL Specifications and Machine Code

P.A. Putro <pavel.putro@ispras.ru>
National Research University Higher School of Economics,
20, Myasnitskaya st., Moscow, 101000 Russia

Abstract. When developing programs in high-level languages, developers have to make assumptions about the correctness of the compiler. However, this may be unacceptable for critical systems. As long as there are no full-fledged formally verified compilers, the author proposes to solve this problem by proving the correctness of the generated machine code by deductive verification. To achieve this goal, it is required to combine the pre- and postcondition specifications with the machine code behavior model. The paper presents an approach how to combine them for the case of C functions without loops. The essence of the approach is to build models, both machine code and its specifications in a single logical language, and use target processor ABI to bind machine registers with the parameters of the high-level function. For the successful implementation of this approach, you have to take a number of measures to ensure the compatibility of the high-level specification model with the machine code behavior model. Such measures include the use of a register type in the high-level specifications and the translation of the pre- and postconditions into the abstract predicates. Also in the paper the choice of logical language for building models is made and justified, the most suitable tools for implementing the approach of merging specifications are selected and the evaluation of the system of deductive verification of machine code built on the basis of the proposed approach is made using test examples obtained by compiling C programs without loops.

Keywords: deductive verification; formal methods; machine code; ACSL

DOI: 10.15514/ISPRAS-2018-30(4)-6

For citation: Putro P.A. Combining ACSL Specifications and Machine Code. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 4, 2018. pp. 95-106. DOI: 10.15514/ISPRAS-2018-30(4)-6

1. Introduction

The paper presents a step forward towards the creation of a tool capable of proving the correctness of machine code based on the formal specification of a function for a high-level language [1]. Such a tool will allow to avoid the assumption about the correctness of the compiler by verification of the generated code regarding specification of source code functionality. The only way in which the correctness analysis of machine code is not necessary is to create a fully formally verified compiler [2].

However, the existing developments in the field of formally verified compilers [3] now do not allow using all the possibilities of existing unverified analogs, for example, GCC [4]. This work is necessary for the implementation of an alternative approach – deductive verification [5] of compiler products, the correctness of which has not been proven. Using this approach will allow you to safely use the already created software.

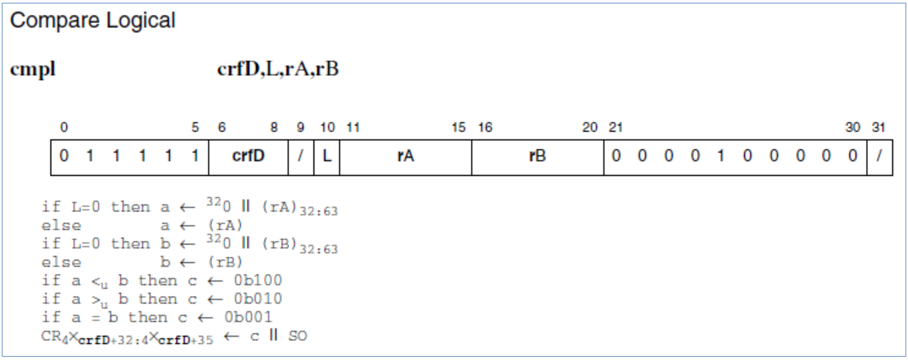
Different approaches to formal specification and building a model of machine code behavior were proposed in different machine code verification projects. Here, the formal specification of a function or a sequence of machine code instructions shows the pre- and postconditions for a function and the behavior model describes mathematical and logical state change formulas. The paper discusses an approach to combining ACSL [6] specifications of the C language with the machine code of the PowerPC e500mc processor obtained by compiling these functions. The choice of the target language is caused by the fact that most high-critical system software like operating system kernels is written in C. While the very high-level languages support a variety of protective mechanisms – such as the prohibition of pointers or checks when casting, the C language is designed for maximum performance by allowing the programmer to interact directly with the memory.

Proof of critical code sections by deductive verification methods can improve the reliability of such systems. In the pursuit of performance, compilers try to make the most of the capabilities of the target processor. Machine code produced by compilers can be extremely difficult for manual verification and specification because the compilation disappears all the information about the names of variables and even the order of execution of commands may be different than in the original program. Only the pre- and postconditions for a particular function remain unchanged. Automatic combination of C-level specifications with the logical model of machine code will allow you to check its correctness in a fully automatic mode.

2. Machine code representation

The specification of machine code instructions in logical languages is a complex and lengthy process. Often, the appearance of the function behavior model specification in this language is very different from that provided in the processor specification. In addition, the lack of special tools makes it difficult to debug such models. To solve these problems, the author proposes to use the NML language, together with the MicroTESK tool [7]. The NML language contains special structures and data types to simplify the modeling of the hardware. The MicroTESK toolset includes universal disassembler of the machine code by the NML language and the NML to SMT-LIB [8] translator.

Fig. 1 shows the cmpl operation specification from the official documentation for PowerPC e500 core family [9] processors and fig. 2 shows its NML version. From here, you can see that the NML language allows you to fully describe processor instructions, including their representation in Assembly language and machine code. In addition, the use of the NML language as the basis for the representation of machine code will allow to reuse all NML models, developed by the MicroTESK development team for the purposes of testing of microprocessors.



Compare Logical

cmpl **crfD,L,rA,rB**

0–5	6–8	9	10	11–15	16–20	21–30	31
0 1 1 1 1 1	crfD	/	L	rA	rB	0 0 0 0 1 0 0 0 0 0	/

```
if L=0 then a ← 32 0 || (rA) 32:63
else        a ← (rA)
if L=0 then b ← 32 0 || (rB) 32:63
else        b ← (rB)
if a <u b then c ← 0b100
if a >u b then c ← 0b010
if a = b then c ← 0b001
CR4×crfD+32:4×crfD+35 ← c || SO
```

Fig. 1. CMPL official specification

```
op cmpl (crfD: CRFD, L: BIT, ra: R, rb: R)
  init = {
    XO_10 = coerce(card(10), 0b0000100000);
    OPCD = coerce(card(6), 0b011111);
  }
  syntax = format("cmpl %d, %d, %s, %s", crfD, L, ra.syntax, rb.syntax)
  image  = format("%6s%3s%1s%1s%5s%5s%10s%1s", OPCD, crfD, L, "0", ra.image, rb.image, XO_10, "0")
  action = {
    if L == coerce(BIT, 0) then
      if ra < rb then
        temp = 0b001;
      endif;
      if ra > rb then
        temp = 0b010;
      endif;
      if ra == rb then
        temp = 0b100;
      endif;
      CR<(coerce(card(5),crfD)*4+2)..(coerce(card(5),crfD)*4)> = coerce(card(3), temp);
      CR<coerce(card(5),crfD)*4+3> = XER_SO;
    endif;
  }
```

Fig. 2. CMPL NML specification

3. ACSL specifications representation

3.1 ACSL specifications translation

As a logical language, in which ACSL specifications will be translated, the author suggests using the WhyML language [10]. The Why3 tool designed to analyze this language allows you to apply many useful transformations and optimizations. It also allows you to translate WhyML code into logical code for many different provers. In addition, the task of translating ACSL specifications into WhyML code has already been solved by the Jessie plugin [11] for Frama-C [12]. In the course of research [1], it was established that the use of the plugin Jessie directly, not suitable for the tasks of machine code analysis.

Jessie plugin makes a number of simplifying assumptions that do not take into account the peculiarities of machine code. Instead, it was decided to take as a basis the unfinished code of jessie3 project [13] – part of the Why3 project. The Jessie3 code has been modified and extended to take into account the peculiarities of machine code. In particular, the language WhyML has been described the type of processor registers. In addition, the algorithm of generating targets for the proof was changed for the subsequent fusion – pre- and postconditions were separated from the function behavior model.

3.2 Using register type for compatibility with machine code

Processor registers can be represented by a limited integer type with an extended set of operations. Operations include signed and unsigned arithmetic, bitwise operations, and memory read operations at the address specified in the register and by offset. To describe all such operations high-level languages, use a variety of different types, as well as a cast operation. However, using different data types will complicate the proof of correctness problem for SMT-solvers. This is especially noticeable in the case of bitwise operations, which are available only for bitvectors in SMT-LIB. Bitvectors cast operations to an integer type are not supported by the latest SMT-LIB [14] standard, and various SMT-solvers offer their own version of the implementation of this operation.

The BitVec type from SMT-LIB is well-suited for describing the type of registers because it contains all the necessary arithmetic and logical sign and unsigned operations. However, the theory of bitvectors at the why3 level does not support all the necessary operations and is built as an unsigned type. Based on the standard theory of bitvectors, the author developed a theory to support the type of processor registers. The theory supports both signed and unsigned integer types and there is ongoing work to add support for pointer arithmetic and memory dereferencing. The driver for CVC4 SMT-solver [15] was updated for translation of the register type to the type BitVec with corresponding mapping of operations.

3.3 Splitting specification and behavior model

To merge machine code, you must separate the pre - and post-conditions from the behavior of the high-level function, which will then be replaced by the behavior of the machine code. To implement this approach, the author uses abstract logical predicates of pre- and postconditions checking. These predicates take as input the parameters of the verification function, and the predicate of the postcondition is also taking its result. Further, by means of axioms predicates are defined by a logical expression in accordance with ACSL specifications. In fig. 3 you can see the predicates for pre- and postconditions are generated based on the ACSL specifications of absolute value function (fig. 4), where usabs_pre – the predicate of a precondition, and usabs_post is a predicate of the postcondition.

```
predicate usabs_post r32 r32

predicate usabs_pre r32

axiom usabs_post_axiom :
  forall n:r32, result:r32.
   usabs_post n result <->
    sge result (of_int 0) /\ (eq result n \/ eq result (sub (of_int 0) n))

axiom usabs_pre_axiom :
  forall n:r32. usabs_pre n <-> slt (neg (of_int 2147483648)) n
```

Fig. 3. WhyML abs logic specification

```
/*@ requires -2147483648 < n;
    ensures \result == n || \result == 0-n;
    ensures \result >= 0;
 */
int abs(int n)
```

Fig. 4. ACSL abs specification

3.4 Replacing proof goal

To facilitate the subsequent merging, the proof goal is substituted during translation of WhyML to SMT-LIB. A new goal for the proof can be described as follows: If the precondition of a function with its arguments is satisfied then the postcondition with the arguments of the function and its result is not satisfied. The negation is used because the SMT-solvers operation specifics – searching for example variable values that will satisfy all restrictions described in SMT-LIB model.

```
;;function argument 1
(declare-const _arg (_ BitVec 32))
;;assign _arg here

;;function result
(declare-const _func_res (_ BitVec 32))
;;assign _func_res here

(assert (usabs_pre
_arg ))

(assert (not (usabs_post
_arg  _func_res)))
```

Fig. 5. Proof goal template

If such an example could not be found then the assumption is incorrect and the predicate of the postcondition is always executed. Therefore, the Expected verdict of the SMT-solver – unsat. It is important to note here that arguments and the result of the function execution are not associated with machine code at this stage – the merge module solves the problem of their binding. Fig. 5 shows SMT-LIB code of goal to prove the correctness of the absolute value function.

3.5 Merging high- and low-level specifications

If you perform all the steps described in the previous sections of this paper, namely, creating an NML model of the machine code and an ACSL to the WhyML translation module, you can perform a merge in two different ways. The first method is the merging at the level of WhyML, and the subsequent translation to SMT-LIB by means of Why3. This approach has a number of advantages, mainly related to Why3 capabilities for WhyML code analysis.

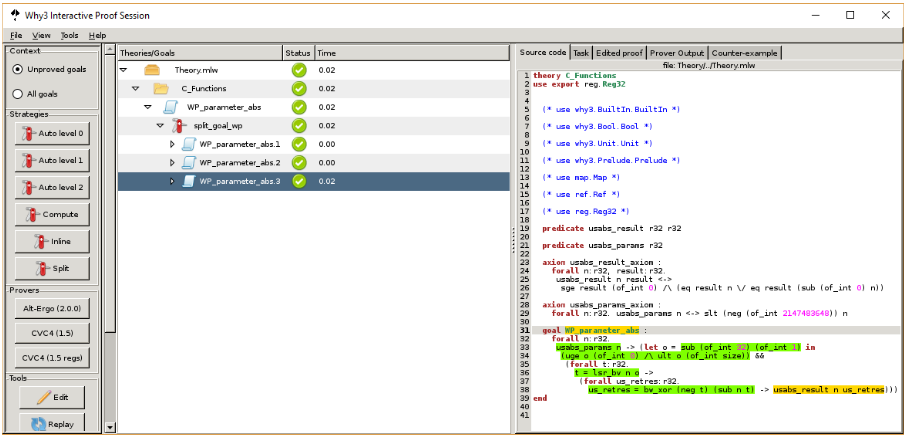


Fig. 6 Why3 IDE

It is worth noting that Why3 IDE (fig. 6), can be used for interactive proof and manual simplifications of verification goals. At the moment the MicroTESK team, with the support of the author, is developing an NML to WhyML translation module. The second approach, as well as the only one implemented at the moment, is merging at the SMT-LIB level. The main advantage of this approach is that the MicroTESK tool has already been implemented NML to SMT-LIB translation module. In addition, the vast majority of operations and data types available in NML have analogs in SMT-LIB.

For example, a set of General-purpose registers is modeled in the NML of the PowerPC processor model as an array of 32-bit registers with a 5-bit index. There is no predefined 5-bit unsigned type in Why3, let alone an array with such an index. However, in SMT-LIB, as in NML, you can manually set the length of BitVec constants. In addition, the translation directly to SMT-LIB allows to avoid unnecessary abstractions that Why3 algorithm for WhyML to SMT-LIB translation can add.

The task of the merge module is to bind together the function arguments and the result of function of high-level language with registers and memory of the model of machine code, and set the environment. Here, the environment refers to machine-specific things, such as the initial value of the stack register or instruction counter. To do this, it is necessary to take into account the specificity of generation SMT-LIB behaviors of the machine code and the specification for the function and specificity of the ABI of architecture.

Next, in fig. 7 we can see binding of the arguments of instructions with the registers for the PowerPC architecture. Developed by the MicroTESK team, generation SMT-LIB by the NML model produces thousands of lines of code. This code can be divided into two main parts: The declaration of all the logical constants needed to describe the behavior model and the description of the state transformation formulas by means of using one assert per machine code instruction and one for every of machine instruction argument.

```
;;function argument 1
(declare-const _arg_1 (_ BitVec 32))
(assert (= _arg_1 (select GPR!1 (_ bv3 5))))

;;function result
(declare-const _func_res (_ BitVec 32))
(assert (= _func_res (select GPR!47 (_ bv3 5))))
```

Fig. 7 Binding function argument and result

4. Evaluation

The developed approach was successfully used to verify the machine code of the absolute value function on the basis of bitwise operations ("Fig. 8"), for which a verdict was obtained, clearly indicating correctness of the function. Tests were also developed to verify the correctness of the implementation of translation of mathematical and logical operations of the ACSL language. Testing of the NML model was done by means of MicroTESK tool.

```
int abs(int n)
{
    int t = (unsigned int) n >> (32-1);
    return (-t) ^ (n-t);
}
```

Fig. 8 Absolute value function

5. Related works

In the why3-avr [16], [17] project, the deductive verification approach is used to prove the correctness of non-loop programs in the assembly language of the AVR microcontroller. The AVR microcontroller used in this study has a fairly simple instruction set that allows you to manually specify the behavior model for each command in the WhyML language, which does not have special means to describe such structures. Also, the model code is described in such a way that allows the programmer to simply copy the function code in the AVR assembly language and add to it a formal specification to get WhyML code for checking the correctness of the function. This approach is especially useful for direct development in a low-level language because the Why3 tool has rich capabilities for transformation and analysis of Why3 code. In addition, the use of Why3 allows converting the WhyML code for proving by various SMT solvers.

However, the program in assembly language is different from compiled machine code that in machine code is a sequence of bytes where there is no all information associated with label names and variables, as well as the formal specification. In addition, machine code does not allow you to abstract from your environment as much as assembly language code. For example, in machine code, indicators such as the address of a function in memory and the value of the stack register at the time of entering the function are important. Also, a high-level formal language specification, such as C, uses various abstractions, such as parameter names and variables, that become unavailable after they are translated into assembly language or machine code. The approaches proposed by the author differ from those described in this project in that they allow using the specification of the high-level language function for analyzing machine code, as well as scaling the supported command system with the help of a specialized modeling language hardware NML.

In the Technical report published by the University of Cambridge Computer Laboratory [18], the HOL4 proof assistant [19] is used for Formal verification of machine-code programs. The paper describes a tool able to verify the machine code for subsets of instructions for popular architectures ARMV4, PowerPC, x86. Behavior model for these instructions was developed by independent developers, so models for both ARM and x86 was designed for HOL4 language [20] [21], and the PowerPC model [22] were manually translated from the Coq language [23] to HOL4.

Here it is worth noting the similarity with the project why3-avr because instructions behavior models were specified manually on unspecialized for such a purpose language. The report terminology uses four levels of abstraction to describe the logical implementation and specification of functions. To obtain a low-level function model (level 2) automatic decompiler translates the machine code (level 1) into recursive functions on the HOL4 language, and also generates their specifications. The use of recursion, in this case, avoids the need to define loop invariants. The user can then focus on interactively proving the properties of the generated function using the HOL4 proof assistant.

For verification, the user also needs to describe the high-level model of the function (level 3), as well as the specification of the function for (level 4). Further, by using relations between levels, user proves that the machine code model complies with the functional specification. In contrast to the interactive HOL4 approach, the approach used in the author's study allows the presence of ACSL specifications to carry out all stages in automatic mode. Also in the author's approach to proving the correctness of machine code is not necessary to have a logical model of the behavior of the function in a high-level language. This degree of automation is achieved including the use of automatic SMT-solvers, in contrast to the interactive proof assistant HOL4. Particularly worth noting is the approach to the translation of programs into recursive functions. The use of high-level language loop invariants at the machine code level is extremely difficult due to the influence of various compiler optimizations. The recursive functions may help to solve these problems.

A number of papers also describe the use of model checking [24] approach for formal verification of machine code. Therefore, in the paper [25] for verification of machine code of the microcontroller Motorola M68hc11 is used Bogor framework [26]. This approach does not imply the presence of function contracts but is based on the use of formally specified behavior models of the system as a whole. As a result, it can be said that the scope of the requirements to be tested varies with the use of deductive verification and model checking.

6. Conclusion

Most of the work that is reviewed specifies the behavior of machine code instructions manually in the logical language. However, in order to simplify and improve the reliability of processor models, the author proposed to describe them in the NML language, designed specifically for such purposes, with the subsequent automatic translation of the model into logical languages. The use of this approach is also facilitated by the presence of a large set of tools in the MicroTESK tool to work with NML, including the NML to SMT-LIB translator. The particularity of ACSL specifications translation to WhyML code, for the case of verification of machine code, such as the need to separate the specification from the behavior model, as well as the importance of the introduction and implementation of the register type.

The observance of such rules and guidelines will allow for automatic merging of function specification and machine code behavior model and thus avoid the need for manual specifying machine code behavior model on the logical language, as required in the project why3-avr. There were proposed two approaches to merge of code specifications and behavior models: at the level of WhyML, and at the level of the SMT-LIB. The first approach allows to use SMT-LIB code generated directly from NML model that help us to avoid extra complexity coming from double translation NML to WhyML and then WhyML to SMT-LIB. The second approach allows to use all the features of the Why3 tool, such as interactive transformations and support of various provers and solvers.

The use of the methods and approaches described in this paper will allow you to fully automate deductive verification of machine code without loops for compliance with the contract specification in ACSL language.

References

[1]. MicroVer – Deductive Verification Tool for Machine Code. Available at: https://forge.ispras.ru/projects/microver, accessed 20.07.2018

[2]. Leroy Xavier. A Formally Verified Compiler Back-end. Journal of Automated Reasoning, vol. 43, issue 4, 2009, pp 363-446

[3]. CompCert – The CompCert C compiler. Available at: compcert.inria.fr, accessed 13-02-2018

[4]. GCC Releases. Available at: http://www.gnu.org/software/gcc/releases.html, accessed 13-02-2018

[5]. Butterfield A., Ngondi G., Kerr A. A Dictionary of Computer Science (ed. 7), Oxford University Press, 2016, 608 p.
[6]. ACSL specification. Available at: http://frama-c.com/acsl.html, accessed 13-02-2018
[7]. Kamkin A., Tatarnikov A. MicroTESK: An ADL-Based Reconfigurable Test Program Generator for Microprocessors. In Proceedings of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE), 2012
[8]. C Barrett, R Sebastiani, S Seshia, and C Tinelli. Satisfiability Modulo Theories. In Handbook of Satisfiability, vol. 185 of Frontiers in Artificial Intelligence and Applications, IOS Press, Feb. 2009, pp. 825–885
[9]. EREF: A Programmer's Reference Manual for Freescale Power Architecture Processors, Rev. 1 (EIS 2.1). Available at: http://cache.freescale.com/files/32bit/doc/ref_manual/EREF_RM.pdf, accessed 13-02-2018
[10]. Filliâtre JC., Paskevich A. Why3 – Where Programs Meet Provers. Lecture Notes in Computer Science, vol. 7792, 2013, pp. 125-128
[11]. M. Mandrykin, A. Khoroshilov. A Memory Model for Deductively Verifying Linux Kernel Modules. Lecture Notes in Computer Sciences. vol. 10742, 2018, pp. 256-275
[12]. Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, Boris Yakobowsk. Frama-c: A Software Analysis Perspective. Formal Aspects of Computing, vol. 27, issue 3, 2015, pp 573–609
[13]. Jessie3 at Why3 source repository. Available at: https://gitlab.inria.fr/why3/why3/tree/master/src/jessie, accessed 12.04.2018.
[14]. Barrett C., Fontaine P., Tinelli C. The SMT-LIB Standard Version 2.6. Available at: http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-r2017-07-18.pdf, accessed 12.04.2018
[15]. Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds. CVC4. Lecture Notes in Computer Science, vol. 6806, 2011, pp. 171-177
[16]. Schoolderman M. Verifying Branch-Free Assembly Code in Why3. Lecture Notes in Computer Science, vol. 10712, 2017, pp. 66-83
[17]. Why3-avr project repository. Availible at: https://gitlab.science.ru.nl/sovereign/why3-avr, accessed 12.04.2018.
[18]. Myreen M.O.: Formal verification of machine-code programs. Ph.D. thesis, University of Cambridge, 2009
[19]. Konrad Slind and Michael Norrish. A brief overview of HOL4. Lecture Notes in Computer Science, vol. 5170, 2008, pp. 28-32
[20]. Anthony Fox. Formal specification and verification of ARM6. Lecture Notes in Computer Science, vol. 2758, 2003, pp 25-40
[21]. Karl Crary and Susmit Sarkar. Foundational certified code in a metalogical framework. Technical Report CMU-CS-03-108, Carnegie Mellon University, 2003.
[22]. Xavier Leroy. Formal certification of a compiler back-end, or: programming a compiler with a proof assistant. In Proc. of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2006, pp. 42-54
[23]. Yves Bertot. A short presentation of Coq. Lecture Notes in Computer Science, vol. 5170, 2008, pp. 12-16
[24]. B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen. Systems and Software Verification: Model-Checking Techniques and Tools. Springer, 2001, 190 p.
[25]. Edelman Joseph R. Machine Code Verification Using the Bogor Framework. Master Thesis, Brigham Young University, 2008
[26]. Bogor framework homepage. Available at: http://bogor.projects.cs.ksu.edu, accessed: 13.02.2018

Совмещение ACSL спецификаций с машинным кодом

П.А. Путро <pavel.putro@ispras.ru>
Национальный исследовательский университет “Высшая школа экономики”,
101000, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. При разработке программ на языках высокого уровня, разработчикам приходится делать предположение о корректности компилятора. Однако это может быть неприемлемо для критически важных систем. Поскольку на данный момент не существует полноценных компиляторов, для которых корректность доказана, автор предлагает решать эту проблему путём доказательства корректности сгенерированного машинного кода методами дедуктивной верификации. Для достижения данной цели необходимо решить ряд задач, одной из которых является слияние модели спецификаций пред- и постусловий с моделью поведения машинного кода. В данной статье представлен подход к проведению слияния спецификаций для случая Си функций без циклов. Суть подхода заключается построении моделей как машинного кода, так и его спецификации на едином логическом языке, и использовании ABI целевого процессора для связывания машинных регистров с параметрами функции высокого уровня. Для успешной реализации такого подхода необходимо предпринять ряд мер по обеспечению совместимости высокоуровневых спецификаций с моделью поведения машинного кода. К таким мерам, в частности, относятся использование типа регистра в высокоуровневых спецификациях, трансляция пред- и постусловий в абстрактные предикаты. Также в статье производится и обосновывается выбор логического языка для построения моделей, выбираются наиболее подходящие инструменты для реализации подхода слияния спецификаций и производится оценка работы системы дедуктивной верификации машинного кода, построенной на основе предложенного подхода, с использованием тестовых примеров полученных путём компиляции Си программ без циклов.

Ключевые слова: дедуктивная верификация; формальные методы; машинный код; ACSL.

DOI: 10.15514/ISPRAS-2018-30(4)-6

Для цитирования: Путро П.А. Совмещение ACSL спецификаций с машинным кодом. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 95-106 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(4)-6

Список литературы

[1]. MicroVer – Deductive Verification Tool for Machine Code, Доступно по ссылке: https://forge.ispras.ru/projects/microver, дата обращения 20.07.2018

[2]. Leroy Xavier. A Formally Verified Compiler Back-end. Journal of Automated Reasoning, vol. 43, issue 4, 2009, pp 363-446

[3]. CompCert – The CompCert C compiler. Доступно по ссылке: compcert.inria.fr, дата обращения 13-02-2018

[4]. GCC Releases. Доступно по ссылке: http://www.gnu.org/software/gcc/releases.html, дата обращения 13-02-2018

[5]. Butterfield A., Ngondi G., Kerr A. A Dictionary of Computer Science (ed. 7), Oxford University Press, 2016, 608 p.

[6]. ACSL specification. Доступно по ссылке: http://frama-c.com/acsl.html, дата обращения 13-02-2018

[7]. Kamkin A., Tatarnikov A. MicroTESK: An ADL-Based Reconfigurable Test Program Generator for Microprocessors. In Proceedings of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE), 2012

[8]. C Barrett, R Sebastiani, S Seshia, and C Tinelli. Satisfiability Modulo Theories. In Handbook of Satisfiability, vol. 185 of Frontiers in Artificial Intelligence and Applications, IOS Press, Feb. 2009, pp. 825–885

[9]. EREF: A Programmer's Reference Manual for Freescale Power Architecture Processors, Rev. 1 (EIS 2.1). Доступно по ссылке: http://cache.freescale.com/files/32bit/doc/ref_manual/EREF_RM.pdf, дата обращения 13-02-2018

[10]. Filliâtre JC., Paskevich A. Why3 – Where Programs Meet Provers. Lecture Notes in Computer Science, vol. 7792, 2013, pp. 125-128

[11]. M. Mandrykin, A. Khoroshilov. A Memory Model for Deductively Verifying Linux Kernel Modules. Lecture Notes in Computer Sciences. vol. 10742, 2018, pp. 256-275

[12]. Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, Boris Yakobowsk. Frama-c: A Software Analysis Perspective. Formal Aspects of Computing, vol. 27, issue 3, 2015, pp 573–609

[13]. Jessie3 at Why3 source repository. Доступно по ссылке: https://gitlab.inria.fr/why3/why3/tree/master/src/jessie, дата обращения 12.04.2018.

[14]. Barrett C., Fontaine P., Tinelli C. The SMT-LIB Standard Version 2.6. Доступно по ссылке: http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-r2017-07-18.pdf, дата обращения 12.04.2018

[15]. Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds. CVC4. Lecture Notes in Computer Science, vol. 6806, 2011, pp. 171-177

[16]. Schoolderman M. Verifying Branch-Free Assembly Code in Why3. Lecture Notes in Computer Science, vol. 10712, 2017, pp. 66-83

[17]. Why3-avr project repository. Availible at: https://gitlab.science.ru.nl/sovereign/why3-avr, дата обращения 12.04.2018.

[18]. Myreen M.O.: Formal verification of machine-code programs. Ph.D. thesis, University of Cambridge, 2009

[19]. Konrad Slind and Michael Norrish. A brief overview of HOL4. Lecture Notes in Computer Science, vol. 5170, 2008, pp. 28-32

[20]. Anthony Fox. Formal specification and verification of ARM6. Lecture Notes in Computer Science, vol. 2758, 2003, pp 25-40

[21]. Karl Crary and Susmit Sarkar. Foundational certified code in a metalogical framework. Technical Report CMU-CS-03-108, Carnegie Mellon University, 2003.

[22]. Xavier Leroy. Formal certification of a compiler back-end, or: programming a compiler with a proof assistant. In Proc. of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2006, pp. 42-54

[23]. Yves Bertot. A short presentation of Coq. Lecture Notes in Computer Science, vol. 5170, 2008, pp. 12-16

[24]. B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen. Systems and Software Verification: Model-Checking Techniques and Tools. Springer, 2001, 190 p.

[25]. Edelman Joseph R. Machine Code Verification Using the Bogor Framework. Master Thesis, Brigham Young University, 2008

[26]. Bogor framework homepage. Доступно по ссылке: http://bogor.projects.cs.ksu.edu, дата обращения 13.02.2018

Prosega/CPN: An Extension of CPN Tools for Automata-based Analysis and System Verification

[1]*J.C. Carrasquel <julio.carrasquel@yahoo.com>*
[2]*A. Morales <ana.morales@ciens.ucv.ve>*
[3]*M.E. Villapol <maria.villapol@aut.ac.nz>*
[1]*La Sapienza University of Rome, Department of Computer, Control, and Management Engineering, Via Ariosto 25, Rome, 00185, Italy*
[2]*Central University of Venezuela, School of Computer Science, Av. Paseo Los Ilustres, Caracas, 1040, Venezuela*
[3]*Auckland University of Technology, School of Engineering, Computer and Mathematical Sciences, 55 Wellesley Street East, Auckland, 1010, New Zealand*

Abstract. The verification and analysis of distributed systems is a task of utmost importance, especially in today's world where many critical services are completely supported by different computer systems. Among the solutions for system modelling and verification, it is particularly useful to combine the usage of different analysis techniques. This also allows the application of the best formalism or technique to different components of a system. The combination of Colored Petri Nets (CPNs) and Automata Theory has proved to be a successful formal technique in the modelling and verification of different distributed systems. In this context, this paper presents Prosega/CPN (*Protocol Sequence Generator and Analyzer*), an extension of CPN Tools for supporting automata-based analysis and verification. The tool implements several operations such as the generation of a minimized deterministic finite-state automaton (FSA) from a CPN's occurrence graph, language generation, and FSA comparison. The solution is supported by the Simulator Extensions feature whose development has been driven by the need of integrating CPN with other formal methods. Prosega/CPN is intended to support a formal verification methodology of communication protocols; however, it may be used in the verification of other systems whose analysis involves the comparison of models at different levels of abstraction. For example, business strategy and business processes. An insightful use case is provided where Prosega/CPN has been used to analyze part of the IEEE 802.16 MAC connection management service specification.

Keywords: formal methods; coloured Petri nets; CPN tools; finite-state automata; protocol verification.

DOI: 10.15514/ISPRAS-2018-30(4)-7

For citation: Carrasquel J.C., Morales A., Villapol M.E. Prosega/CPN: An extension of CPN Tools for Automata-based Analysis and System Verification. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 4, 2018. pp. 107-128. DOI: 10.15514/ISPRAS-2018-30(4)-7

1. Introduction

The verification of distributed systems and the assurance of their correctness is a task of utmost importance; specially in today's world where many critical services are completely supported by computer technologies. Among the solutions for system modelling and verification, Petri Nets [1] play a major role since its capability of graphically visualize systems, and for maintaining the formal rigor, so it allows to perform a convenient analysis of the behavioral properties of a system. Thus, the formalism of Petri Nets has been extended to other models in order to enrich their expressiveness and practicability. Particularly, we consider Coloured Petri Nets (CPNs) [2] where data types (*colors*) may be associated to net elements. CPN Tools [3] is a consolidated software tool for editing, simulating, and analyzing CPN models.

However, when dealing with a higher complexity of the system, it may be useful to combine the usage of different analysis techniques. This also allows the application of the best formalism or technique to different components of a system. In the context of Colored Petri Nets, the last version of CPN Tools includes the Simulator Extensions whose development has been driven by the need of integrating CPN with other formal methods [4]. In particular, we consider the integration of CPNs and Finite-state Automata (FSA) which has been proved to be useful for the validation of different protocols and communication systems [5] [6] [7].

For instance, given a CPN's occurrence graph (OG), the arcs through a path in the OG may be seen as the sequence of service primitives that a user (i.e. another system entity in a higher layer) invokes in order to request some action by a service provider. The nodes in the OG may be considered as changes of state in the system due to the services invocations. Finally, some nodes of the OG may represent halt states, meaning the termination of a specific process. Hence, the OG can be seen as a FSA, which can be analyzed using well-known algorithms and theorems.

There are several tools for building, combining, optimizing, and searching Finite-state Automata. However, in order to apply them for analyzing CPNs and occurrence graphs, these ones must be converted into FSA specific formats (i.e. see [5] [6]). Using several tools may complicate the verification process.

Thereby, we developed a solution called Prosega/CPN (*Protocol Sequence Generator and Analyzer*). The tool aims to bridge conveniently the formalism of CPNs with Finite-state Automata, taking advantage of the Simulator Extensions feature in CPN Tools. Thus, the software provides a mechanism for transforming a CPN's occurrence graph into a minimized deterministic FSA as well as other operations for language generation and FSA comparison. Prosega/CPN has been conceived to support the protocol verification methodology proposed by Billington [8]. However, the tool may be useful to support the verification of other systems whose strategy may involve the usage of FSAs, or the comparison of models at different levels of abstraction; for example, business strategy and business processes.

The remainder of this paper is structured as follows. Section 2 introduces the literature related to our work. Section 3 presents some formal definitions for understanding the models managed by Prosega/CPN. Sections 4 and 5 describe the tool functionalities and architecture respectively. Section 6 describes a use case where the tool has been used to analyze part of the IEEE 802.16 MAC connection management service specification. Finally, Section 7 presents the conclusions.

2. Related work

Prosega/CPN has been developed within the context of system verification through the formalism of Coloured Petri Nets (CPNs) and Finite-state Automata (FSA). The tool has been conveniently developed as an extension of CPN Tools [3] since it performs several operations on FSAs generated from a CPN model. i.e. the reduction of a CPN's occurrence graph into a FSA. Hence, through the development of Prosega/CPN we have been focused in three topics within the literature:

- Works dealing with the development of extensions for CPN Tools [4] [9] [10] [11].
- Tools and other solutions for the analysis and manipulation of FSA [12] [13] [14] [15] [16].
- Works proposing a system verification methodology using CPNs and FSA, and the use cases in which it has been applied [5] [6] [7] [8] [17], and other scenarios where both formalisms have been used together [18] [19] [20].

CPN tools has a history for communicating with external solutions; its architecture provides a set of communication primitives for connecting external software to the CPN simulator engine. As an initial effort, it was developed Comms/CPN [9], a library for Java and C/C++ which makes it possible for CPN Tools to communicate based on TCP/IP with external application and processes. The BRITNeY Suite [10] is other solution which provides model visualizations in an external tool, and more recently Access/CPN [11] that provides a channel to interact with the CPN Tools simulator engine from external Java programs. However, while these previous tools have made it easy to *interact* with CPN Tools, they have not made it possible to *extend* the software. Thereby, it was developed the Simulator Extensions [4] feature included in the last version of CPN Tools. This component provides a mechanism for adding new functionalities within the CPN Tools Graphical User Interface (GUI), thereby allowing integrating other related formalisms with CPN models; as a result, it has been possible to handle other models in the tool such as low-level Place/Transition nets, Declare models, and drawing message sequence charts from model executions [4].

On the other hand, Finite-state Automata (FSA) have been used in a much wider spectrum of fields than CPNs; as an important tool for FSA manipulation we highlight the FSM Library from AT&T Labs [12] which is a collection of Unix software tools for creating and manipulating finite-state machines. Despite the library is quite general purpose, it was designed for speech processing applications such as speech recognition/synthesis; FSM Library was used as well in previous works regarding the verification of communication systems based on CPNs and automata [5] [6]. Some of the researchers of the AT&T FSM project developed later an enhanced version called OpenFST [13], which is an open-source alternative that also allows to construct finite-state transducers, and it provides a C++ template library. Within the range of tool solutions for FSA manipulation, we may also find Foma [14], the FAdo project [15] and the specialized pedagogical tool JFLAP [16] among many others.

Bridging CPNs and FSA may be useful for verification of systems of very high complexity. In particular, Billington [8] proposed a CPN and FSA approach for the verification communication systems that has proven to be successful; namely, in the verification of the Resource Reservation Protocol (RSVP) [5], the Wireless Application Protocol (WAP) [6], the Transmission Control Protocol (TCP) [7], and the Internet Open Trading Protocol (IOTP) [17], among other cases. Between other domains in which both formalisms have been applied together we may find the verification of web-services composition [19] [20] or vehicular traffic control systems [18], just to mention a few.

3. Formal Definitions

This section presents some formal definitions of the models and data structures that are manipulated through the functionalities of CPN Tools and Prosega/CPN. In particular, it is formulated how it can be derived an occurrence graph (OG) from a CPN model, and afterwards is explained how can it be generated a Finite-state Automaton (FSA) from a CPN's occurrence graph. The following formulations are based in the work done in [8]. Albeit CPNs are managed in this work; for the formal definition it has been rather convenient to generalize the type into a High-level Petri Net (i.e. for proving further theorems regarding the relationship between an OG and a FSA as described in [8]). Hence, we firstly take the definition of a High-level Petri net (HLPN) [21].

Definition 1. *A High-level Petri Net is a structure of the form* $HLPN = (P, T, D; Type, Pre, Post, m_0)$ *where:*

- P *is a finite set of Places;*
- T *is a finite set of Transitions such that* $P \cap T = \emptyset$
- D *is a non-empty finite set of non-empty domains where each element of* D *is called a type.*
- $Type: P \cup T \rightarrow D$ *is a function used to assign types to places, and to determine transition modes.*
- $m_0 \in \mu PLACE$ *is a multi-set called the initial marking of the net such that* $\mu PLACE$ *is a set of all possible multi-sets of* $PLACE$
- $Pre, Post: TM \rightarrow \mu PLACE$ *are the pre and post mappings with*
    - $TM = \{(t, m) | t \in T, m \in Type(t)\}$ *the set of transition modes.*
    - $PLACE = \{(p, g) | p \in P, g \in Type(p)\}$ *the set of elementary places.*

For the analysis of a High-level Petri net it is generated an occurrence graph (OG). We consider that an OG can be defined as a labelled and rooted directed graph, where the nodes of the graph represent *markings* of the Petri Net, and the directed arcs represent the *transition modes* (or binding elements [2]) that can *occur* in all executions from the initial marking. On the other hand, the root of the graph refers to a node, which is considered as the initial state. In addition, the arcs of an OG may be labelled by the transition modes. Thus, we start by defining a labelled and rooted directed graph, and then we give the definition of an OG associated to a HLPN.

Definition 2. *A labelled directed graph, with v_0 as the root node, is a triple $G = (V, L, E)$ where:*

- *V is a finite set of vertices or nodes; $v_0 \in V$ represents the root or initial node.*
- *L is a set of labels;*
- *$E \subseteq V \times V$ is a set of labelled directed edges.*

Definition 3. *An occurrence graph of a HLPN with an initial marking v_0 is a labelled and rooted directed graph $OG = (V, TM, A)$ where*

- *V is a finite set of vertices or nodes reachable from m_0 (the reachability set); $m_0 \in V$ represents the initial marking (root node);*
- *TM is the set of transition modes of the HLPN;*
- *$A = \left\{(m, tm, m') \in V \times TM \times V' | m \overset{tm}{\rightarrow} m'\right\}$ is the set of arcs (directed edges) labelled by transition modes.*

Remark. *$m \overset{tm}{\rightarrow} m'$ indicates the ocurrence of a transition mode tm $\in$ TM in a marking m which results in a new marking m'*

However, when we are only interested in the transition names, then the arcs of the OG are just labelled with such transitions names rather than the transition modes (binding elements). For example this is useful when it is just required to understand which user observable events (service primitives) may lead from a state of the system to another one; instead of transition modes which involve the parameters binded to such events. In addition, when we are also interested in the identification of the markings for the nodes of the OG, rather than the marking details, we introduce an injection $I: [m_0] \rightarrow \mathbb{N}$ such that this function maps the set of reachable markings from m_0 (denoted as $[m_0]$) into the set of natural numbers. Giving the described abstractions for transitions and markings, we consider the definition of an abstract OG.

Definition 4. *An abstract OG of a HLPN with an initial marking m_0 is a labelled and rooted directed graph $OG = (V, T, A)$ where*

- *$V = \{I(m) | m \in [m_0]\}$ is the set of nodes;*
- *$I(m_0) \in V$ represents the root or initial node.*
- *T is the set of transitions of the HLPN ;*
- *$A = \left\{\big(I(m), t, I(m')\big) \in V \times T \times V' | m \overset{(t,m) \in TM}{\rightarrow} m'\right\}$ is the set of arcs labelled by transition.*

We point out that the abstract occurrence graph OG defined above is *finite*. i.e. It has a finite number of states. Indeed this is an important fact when dealing with real scenarios. This means that the corresponding Petri Net must be a bounded net [1], and hence a preliminary boundedness analysis on the Petri Net is performed. Finally, it is presented a mapping from an abstract OG (Definition 4) into a Finite-state Automaton FSA. We define a function $Prim: T \rightarrow SP \cup \{\varepsilon\}$ that maps each transition of the HLPN to either an identifier name (i.e. an user observable event or service primitive name), or to an *epsilon* (i.e. an empty move); *SP* is the set of identifiers (for the user observable events or service primitive names) for the system that we are describing.

Definition 5. *A Given an abstract occurrence graph* $OG = (V, T, A)$ *it is derived the corresponding Finite-state Automaton* $FSA = (V, SP, A_{SP}, v_0, F)$ *where*

- V *is the set of nodes of the abstract OG (the states of the FSA);*
- SP *is the set of identifiers (for the user observable events or service primitive names) of the system of interest (the alphabet of FSA);*
- $A_{SP} = \{(v, Prim(t), v') | (v, t, v') \in A\}$ *is the set of transitions labelled by elements of SP or epsilons (the transition relation of the FSA);*
- v_0 *corresponds to the abstract initial marking (initial state of the FSA).*
- $F \subseteq V$ *the set of final (acceptance) states.*

Prosega/CPN performs the conversion of an OG as described in Definition 4 into a FSA as described in Definition 5. Moreover, this mapping between OG and the FSA allows the tool conveniently manage the generation of the language and the comparison between other FSAs.

4. Functionalities

Prosega/CPN is an extension in CPN Tools. Thus, the user interacts with the application using a Graphical User Interface (GUI) through a tool palette added to CPN Tools (see fig. 1) - available under the Tool box entry [3]. The tool supports the generation of a minimized deterministic Finite-state Automaton (FSA) derived from the CPN’s occurrence graph, the language generation, and the comparison between two different FSAs. We proceed to explain these functionalities in detail.

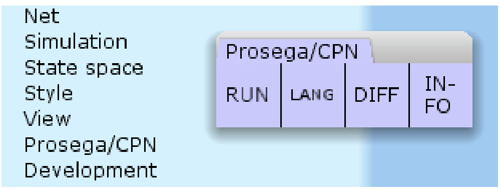


Fig. 1. Tool palette of Prosega/CPN

4.1 FSA Generation

Once the occurrence graph (OG) from a CPN model is generated using the CPN Tools simulator [3], its associated Finite-state Automaton (FSA) can be generated and reduced using the RUN tool (see Fig. 1). To this aim, the following steps are performed: getting the transitions, and dead markings of the OG, assigning identifiers to transitions (i.e. constructing the mapping *Prim* defined in Section 3), reducing the FSA, and displaying the results. Here, we consider the structure of an abstract OG where the nodes are identified by numbers, which represent the markings and the arcs are just labelled with the transitions rather than the binding elements (see Definition 4).

Firstly, the tool communicates with the CPN Tools simulator in order to obtain all the transitions and the dead markings (see Section 5). The user interacts with the Prosega/CPN GUI to assign identifiers (corresponding to user observable events or service primitive names) to the model transitions (i.e. mapping elements from a set *SP*). The character 0 is considered as an epsilon (ε). Hence, any transition assigned with 0 is considered an epsilon transition (or empty move). Then, the user chooses the set of terminal states *F* for the FSA. which may include nodes representing the dead markings or other nodes in the OG. Thereby, it is obtained a FSA in line with Definition 5.

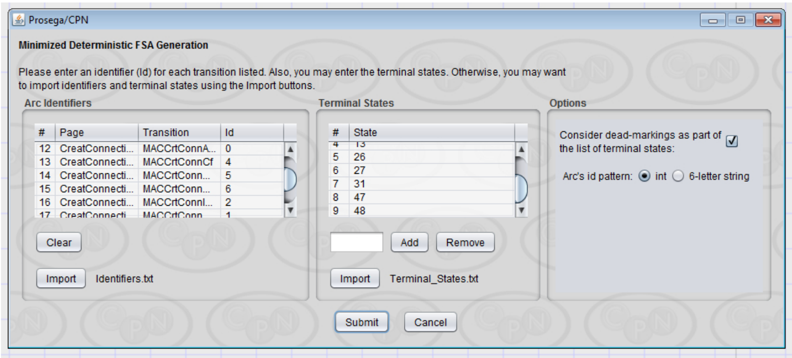


Fig. 2. Intial Prosega/CPN in terface where the user can assign Ids to transitions and enter terminal states

For instance, fig. 2 shows the Prosega/CPN interface which supports the described operation. In particular, it is defined a FSA given a CPN's occurrence graph extracted from the use case in Section 6. The user assigns identifiers for the CPN transitions. For example, the identifier 1 to the transition *MACCrtConnReq*, which is in the CPN model page *CreatConnection*. Later, the user chooses the following nodes of the OG as terminal states: 1, 7, 8, 13, 26, 27, 31, 48 (some are not displayed in the figure due to window size limitation). Afterwards, the modelled FSA is reduced by following the algorithm described in [22], which consists in performing the following operations over a FSA:

- removal of *epsilon* transitions (remove empties);
- removal of non-determinism (determinization);
- reduction by identifying and merging equivalent states (minimization).

The algorithm produces as output a reduced deterministic FSA with a minimal number of states that is equivalent to the input automata. Finally, an interface showing the results of the FSA reduction is displayed to the user as shown in fig. 3.

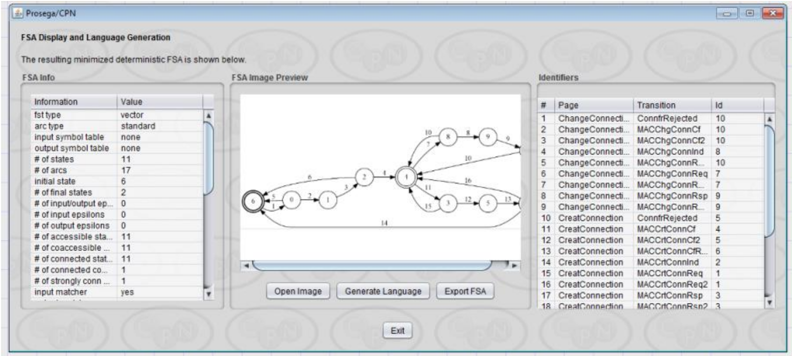


Fig. 3. Interface showing the results of the FSA reduction process

The interface shows general information about the reduced FSA (FSA Info), such as initial state and number of arcs, which may be relevant for the FSA analysis. It also includes a graphical representation of the FSA (FSA Image Preview), and the established mapping between the identification numbers/names assigned by the user and the transition names, which may be useful for debugging and verification of the model.

4.2 Language generation

The language accepted by a FSA can be generated by using either the LANG tool in Fig. 1 or the Generate Language button in Fig. 3. The interface shown in Fig. 4 is displayed to the user after it clicks on the LANG tool. Then the user can choose both the FSA, in plain text or in the compiled format [13], for which the language will be generated and the corresponding symbol table file—for mapping the arc inscriptions with the symbols selected by the user.

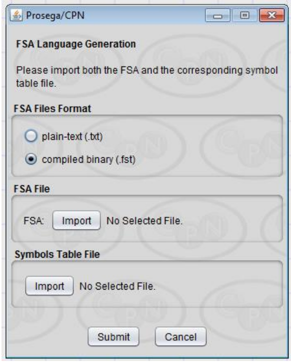


Fig. 4. Language generation interface

The language generator module generates the language *L* of the FSA by extension; if *L* is finite, the whole sequences are printed; otherwise a subset of the language, *L'* ⊆ *L* is generated, as illustrated in fig. 5. In particular, *L'* is a set of symbol sequences whose symbols belong to different arcs in the FSA. Notice that some arcs of the FSA may be labelled with the same symbol. However, in the generation of each sequence, each arc of the FSA is visited just once.

Indeed, for generating each sequence accepted by the automaton it was developed an algorithm based on iterative Depth-first Search (DFS), which was implemented in the language generator component of Prosega/CPN (as mentioned in Section 5). This component performs DFS between the initial state of the FSA, to each of the halt states. Hence, the symbols of the arcs visited through the path from the initial state to a specific halt state are printed, thereby representing a sequence accepted by the automaton. In addition, this module supports a generator of random sequences of the language symbols, as shown in Fig. 6, which may be useful when the language is infinite. For example, in Fig. 5 and 6, we can see the following sequence of language symbols: 1, 5 which corresponds to the sequence of actions (transitions): *MACCrtConnReq*, *MACCrtConnCf2* (as shown in the interface in fig. 4, where the user assigned an Id (language symbol) to each transition).

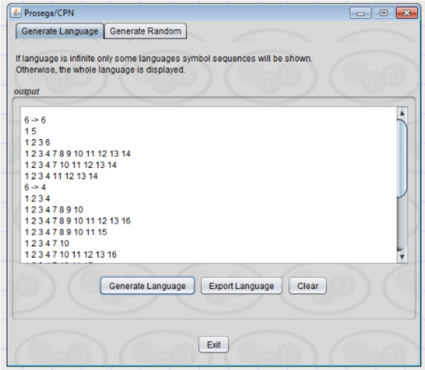


Fig. 5. Interface showing part of the language accepted by the FSA of Fig. 3

In particular, for generating each random sequence it is computed a random walk in the FSA from the initial state to any of the halt states. Whenever a halt state is visited, the walk will be terminated with a probability $p/100$ s.t $0 < p \leq 100$, and the sequence of symbols, which were collected throughout the visited path will be printed. Thus, in the Generate Random interface (fig. 6), the user can manipulate the average size of the randomly generated sequences of language symbols by entering the halt-rate parameter value p. Therefore, if the value p is close to 0, the number of language symbols in each sequence may be big, while if p is close to 100, then the number of language symbols in each sequence may be small, thereby determining the length of each sequence. i.e. since the halt-rate parameter value in fig. 6 is 55, in that case the sizes of the sequences are medium.

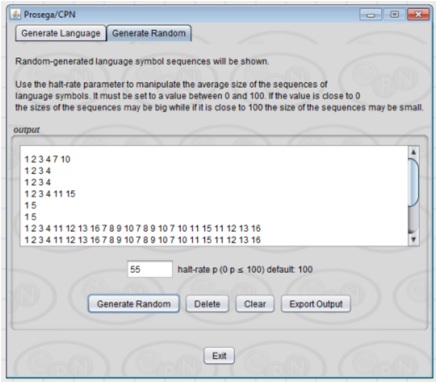


Fig. 6. Interface showing some randomly generated sequences of language symbols

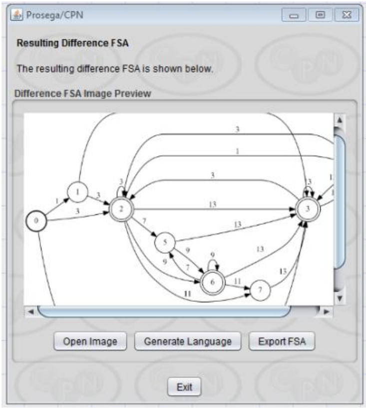


Fig 7. The interface shows the resulting difference FSA given two automata as parameters

4.3 FSA Difference

The user can use the DIFF tool to calculate the difference between two automata, F A and F B. This functionality, whose output interface is illustrated in fig. 7, generates a new automaton F C which only accepts the sequence of symbols accepted by the first automaton F A , and that are not accepted by the second one F B . In particular, F B must be an epsilon-free, deterministic finite automaton. This is useful to understand the sequences of languages symbols in which may differ two models; in this sense, as seen in fig. 7, this functionality allows to generating the language of F C for getting such sequences in which may differ two models.

5. Architecture

Prosega/CPN is implemented in Java programing language, so we use the new feature in CPN Tools 4 called Simulator Extensions [4] to add the software functionalities. Fig. 8 shows the software architecture, which illustrates the relation ship among all the components of our tool, CPN Tools and the third-party components. Communication between the CPN Tools GUI and the simulator, and between the simulator and the Simulator Extensions is supported by the BIS (Boolean - Integer - String) protocol. Each protocol message is encoded using a number of booleans, integers, and strings as explained in [23]. In order to facilitate the development of Prosega/CPN we use some third-party libraries, which implement many of the functions to manage and display the automata.

In particular, we utilize OpenFST [13] [24] for FSA reduction and FSA difference, and Graphviz [25] for drawing the automata. On the other hand, we wrote the code for language generation (fsm2language) in C programming language [26]. The fsm2language implements the procedures for language generation and the computation of random sequences accepted by a FSA that were described in Section 4. The bridge between the fsm2language component and the Prosega/CPN tool is supported by JNI (Java Native Interface), which enables a Java program to call native libraries written in C/C++ programming language.

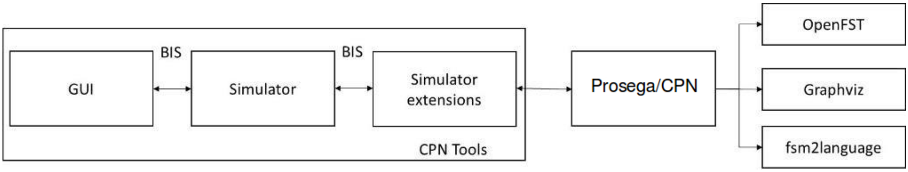


Fig. 8. Prosega/CPN Architecture

6. Use Case

The IEEE 802.16 standard [27] is responsible for specifying and describing the air interface of Broadband Wireless Access Systems (BWA), and point-multipoint fixed/mobile wireless metropolitan area network. The standard is limited to the description of the Medium Access Control (MAC) and physical (PHY) layers. In overall, IEEE 802.16 provides great benefits for providing mass broadband wireless connectivity, allowing user mobility, mesh-mode network support, and even has been thought as an alternative for Internet-of-Things deployments. However, due to its inherent complexity, there are several parts of the specification that turn out to be ambiguous, difficult to understand and imprecise. In this context, Morales et al. [28] [29] has contributed establishing a formal model for a module of IEEE 802.16. In particular, it developed a formal verification of the MAC connection management service specification. To this aim, the Prosega/CPN tool has been used in conjunction with the Billington's protocol verification methodology [8]. Fig. 9 illustrates the steps of the methodology; we proceed to explain such steps, and how they have been applied within our use case using CPN Tools and Prosega/CPN.

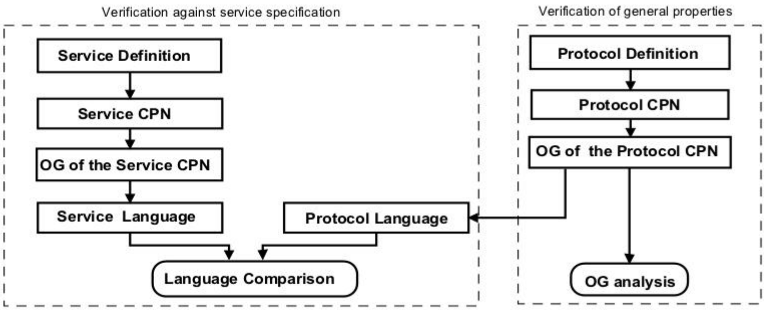


Fig. 9. Steps within the protocol verification methodology proposed in [8].

6.1 Service Definition

In fig. 9, the dashed box in the left represents the first step which consists in modelling the service specification of the system, and to define the services that it aims to provide (either to a higher layer or to another system entity). In the scenario of the IEEE 802.16 MAC layer, the service specification consists in a set of service primitives that the MAC sub-layer, responsible for connection management procedures, provides to the sub-layer on top of it. Each of these primitives correspond to one of the following procedures: The establishment of a connection between communication peers, the connection maintenance (i.e. management of the dynamic network resources) and the termination of the connection by any of the communication peers.

6.2 Service CPN and OG

Using CPN Tools, it is created the CPN model of the service specification. fig. 10 presents the CPN main page which shows a top view of the model [2]. This top module is linked with the pages that model the service primitives that correspond to the establishment, maintenance, and termination of a connection through the transitions *CreatConnection*, *ChangeConnection*, and *TerminateConnection* respectively. Each of these pages of the model can be checked in [28]. Afterwards, it is generated the CPN's occurrence graph (OG), shown in fig. 11, which is the input for the FSA reduction feature of Prosega/CPN.

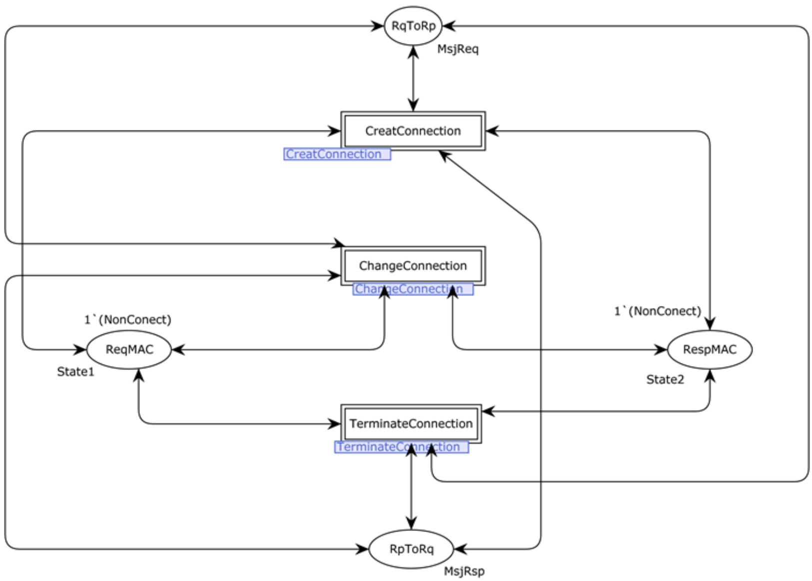


Fig. 10. CPN model representing the hierarchical view for the processes of creation, change and termination of connections between peer MAC entities in the IEEE 802.16 service specification

6.3 FSA Reduction

Once the service OG is generated, it is modelled as a FSA in line with Definitions 4 and 5. To this aim, it is used the RUN tool of Prosega/CPN for converting the OG into a FSA (as presented in fig. 2). For each transition of the CPN model, it is assigned a number value which represents the associated service primitive identifier (Id) (resembling the function *Prim* described in Section 3). Transitions that are considered as empty moves (or internal events) are labelled with 0 (*epsilon* transitions). Later, there are assigned the terminal states. The assignation performed between all the model transitions and the service primitive identifiers as well as the decision of the terminal states can be fully checked in [28]. Afterwards, the FSA is minimized following the procedure explained in Section 4. Fig. 12 presents the minimized deterministic FSA (exported from the output/analysis interface of the RUN tool previously presented in Fig. 3.

Table. 1. Service primitivies on the IEEE 802.16 MAC Layer and their corresponding identification number [22]

Service Primitive	Id
MAC_CREAT_CONNECTION.Request	1
MAC_CREAT_CONNECTION.Indication	2
MAC_CREAT_CONNECTION.Response	3
MAC_CREAT_CONNECTION.Confirmation	4, 5, 6
MAC_CHANGE_CONNECTION.Request	7
MAC_CHANGE_CONNECTION.Indication	8
MAC_CHANGE_CONNECTION.Response	9
MAC_CHANGE_CONNECTION.Confirmation	10
MAC_TERMINATE_CONNECTION.Request	11
MAC_TERMINATE_CONNECTION.Indication	12
MAC_TERMINATE_CONNECTION.Response	13
MAC_TERMINATE_CONNECTION.Confirmation	14, 15, 16

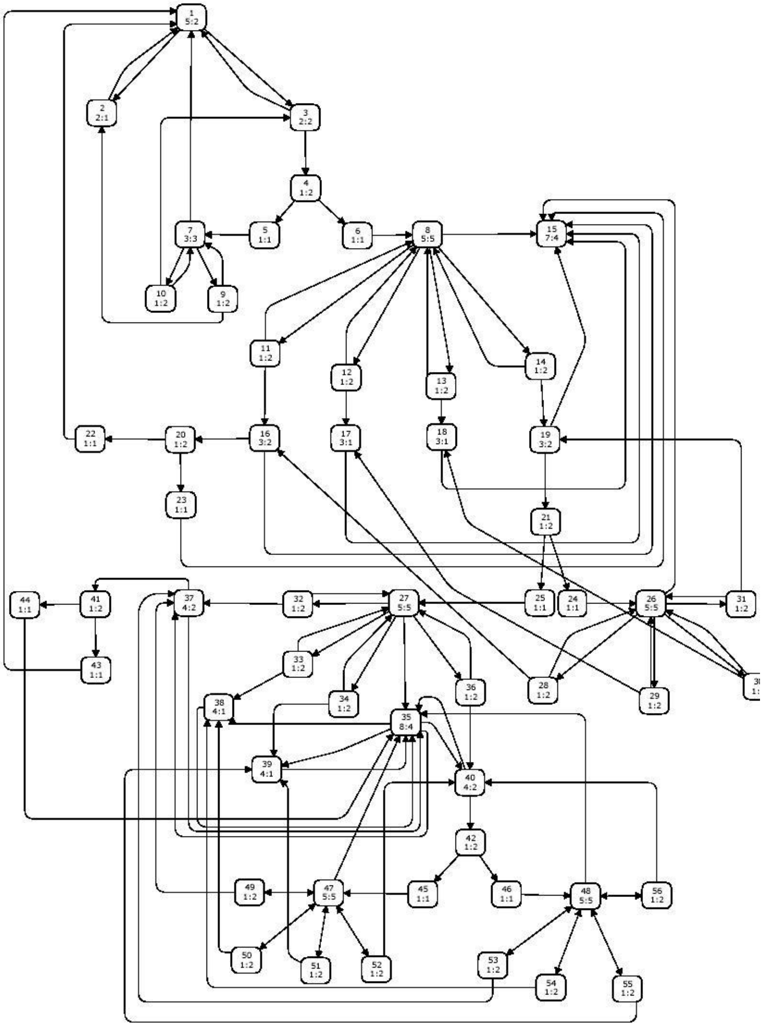


Fig. 11. OG of the CPN model representing the IEEE 802.16 MAC connection management service specification.

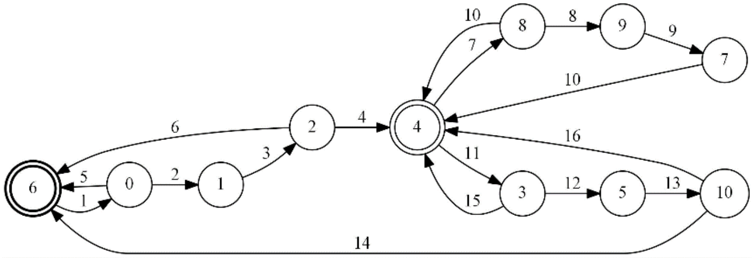


Fig. 12. Minimized deterministic FSA generated from the OG illustrated in fig. 11

6.4 Language Generation

The service language (the set of sequences of service primitives) is generated using Prosega/CPN as explained in Section 4 —utilizing FSA minimization (RUN tool) and FSA language generation (LANG tool). Fig. 5 presented some sequences that are accepted by the FSA. In addition, Table. 1 shows the identifier selected for each primitive service [28]. For example, the sequence of language symbols 1, 2, 3, 4, 7, 8, 9, 10 represent the service primitives invoked by the protocol entity in top of the MAC for the successful establishment and maintenance (change of a communication resource) of the connection. In overall, the minimized FSA generated by Prosega/CPN provides a compact description of the possible sequences of service primitives, and allows to remove complexity from the model, which allows the language to present a clear specification of the service that the system provides.

6.5 Further Steps

The second part of the methodology (dashed box in the right of Fig. 9) concerns to the modelling of the protocol, and its comparison against the service specification through language equivalence. These further steps are still in progress within the research work [28]. The modelling of the protocol consists in constructing the CPN model, which describes the protocol procedures which are performed when a service primitive is invoked by a higher entity of the system. Later, it is generated the OG associated to this CPN model. On the one hand, behavioral properties of the protocol may be analyzed through the OG. On the other hand, the OG may be reduced into a minimized deterministic FSA. i.e. using again the RUN tool of Prosega/CPN.

Then, the FSA of the service specification may be compared with the FSA of the protocol. i.e. using the DIFF function of Prosega/CPN – see fig. 7. Finally, the language of the difference FSA may be generated in order to determine language equivalence between the service and the protocol. Thus, we can determine the sequences of service primitives, which are in the protocol specification but are not in the service specification. It is important to know if the service specification meets the protocol specification, since it is not desirable to have a service requirement from the service user which cannot be met by the protocol. In addition, it may not be wanted a service provided by the protocol which actually it is never required by the user.

7. Conclusion

This work has presented Prosega/CPN. The tool is an extension of CPN Tools for supporting several operations for FSA-based analysis and system verification. The tool provides a feature for generating a minimized deterministic Finite-state Automaton (FSA) from a CPN's occurrence graph (OG). It includes as well operations for language generation, and for automata comparison. These functionalities are supported taking advantage of consolidated third-party components such as OpenFST and Graphviz. In addition, we developed a module for language generation.

Prosega/CPN has been integrated within the CPN Tools GUI using the Simulation Extensions (new feature in the last version of CPN Tools) component whose development has been driven by the demand of many research works to suitably integrate Colored Petri Nets with other formalisms [4]. In particular, the integration between CPNs and FSA was not existing within CPN Tools, and the application of this multi-formalism strategy has shown its merits in many published papers, specially from the domain of protocol verification.

Furthermore, other works may be benefited from this FSA-based verification; for example, as presented in our use case, the analysis of an equivalent reduced FSA provides a compact and clear description of the possible user observable events (service primitive calls) rather than to deal with the analysis of the OG, thereby allowing to reduce the time complexity when it may be required to check the behavioral properties of the system through the FSA.

As future work, the tool will keep providing support within the further steps of the formal verification work of the IEEE 802.16 standard, regarding to the MAC connection management procedures. On the other hand, as another further direction for the tool enhancement, the tool has been thought to be tested in other domains; indeed, as it has been stated, Prosega/CPN can be used in other cases where FSA may be required, and within the verification of other systems whose analysis may involve the comparison of models at different levels of abstraction.

This future work on other use cases will be able to keep maturing the tool. i.e. integrating new operations/features for automata manipulation, and testing the tool performance in terms of scalability, among other key facts. In addition, it has been considered to keep exploiting more capabilities offered by the Simulator Extensions channel; for example, to be able draw and manually edit a FSA in the CPN Tools canvas, instead of only using the Graphviz support for automata drawing.

References

[1]. T. Murata. Petri nets: Properties, analysis and applications. Proceedings of the IEEE, vol. 77, no. 4, April 1989, pp. 541–580

[2]. K. Jensen and L. M. Kristensen. Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Berlin, Heidelberg: Springer-Verlag, 2009

[3]. CPN Tools – A tool for editing, simulating, and analyzing Coloured Petri Nets. Available at: http://www.cpntools.org/, accessed: 20.06.2018

[4]. M. Westergaard. CPN Tools 4: Multi-formalism and Extensibility. In Application and Theory of Petri Nets and Concurrency. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 400–409

[5]. M. E. Villapol. Modelling and Analysis of the Resource Reservation Protocol Using Coloured Petri Nets. Ph.D. dissertation, University of South Australia, Australia, December 2003

[6]. S. Gordon, L. M. Kristensen, and J. Billington. Verification of a Revised WAP Wireless Transaction Protocol, In Application and Theory of Petri Nets and Concurrency. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 182–202

[7]. B. Han. Formal Specification of the TCP Service and Verification of TCP Connection Management. Ph.D. dissertation, University of South Australia, Australia, April 2004

[8]. J. Billington, G. E. Gallasch, and B. Han. A Coloured Petri Net Approach to Protocol Verification. Berlin, Heidelberg: Springer-Verlag, 2004, pp. 210–290.

[9]. G. Gallasch and L. M. Kristensen. Comms/CPN: A Communication Infrastructure for External Communication with Design/CPN. In Proc. of the Third Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, DAIMI PB-554, pages 75–91

[10]. M. Westergaard and K. B. Lassen. The BRITNeY Suite Animation Tool. In Applications and Theory of Petri Nets and Concurrency. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 431–440

[11]. M. Westergaard. Access/CPN 2.0: A High-Level Interface to Coloured Petri Net Models. In Application and Theory of Petri Nets and Con- currency. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 328–337

[12]. AT&T Researchers – Inventing the Science Behind the Service. Available at: http: //www.research.att.com/evergreen/portfolio/, accessed: 20.06.2018

[13]. OpenFST Library. Available at: http://www.openfst.org/twiki/bin/view/FST/WebHome, accessed: 20.06.2018

[14]. M. Hulden. Foma: A Finite-state Compiler and Library. In Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: Demonstrations Session. Stroudsburg, PA, USA: Association for Computational Linguistics, 2009, pp. 29–32

[15]. A. Almeida, M. Almeida, J. Alves, N. Moreira, and R. Reis. FAdo and GUItar: Tools for Automata Manipulation and Visualization. In Implementation and Application of Automata. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 65–74

[16]. S. H. Rodger. JFLAP: An Interactive Formal Languages and Automata Package. USA: Jones and Bartlett Publishers, Inc., 2006

[17]. C. Ouyang and J. Billington. Formal Analysis of the Internet Open Trading Protocol. In Applying Formal Methods: Testing, Performance, and M/E-Commerce. Berlin, Heidelberg: Springer-Verlag, 2004, pp. 1–15

[18]. S. Barzegar, M. Davoudpour, M. R. Meybodi, A. Sadeghian, and M. Tirandazian. Traffic Signal Control with Adaptive Fuzzy Coloured Petri Net Based on Learning

Automata. In Annual Meeting of the North American Fuzzy Information Processing Society, July 2010, pp. 1–8

[19]. N. Danapaquiame, E. Ilavarasan, N. Kumar, and S. K. Dwivedi. Ratification strategy for web service composition using CPN: A survey. In Proc. of the IEEE International Conference on Computational Intelligence and Computing Research, December 2013, pp. 1–4

[20]. J. Zhu, K. Zhang, and G. Zhang. Verifying Web Services Composition based on LTL and colored Petri Net. In Proc. of the 6th International Conference on Computer Science Education, August 2011, pp. 1127–1130

[21]. ISO/IEC. High-level Petri Nets – Part 1: Concepts, Definitions and Graphical Notation. Software and Systems Engineering, ISO/IEC FDIS 15909-1. Final Draft International.

[22]. W. A. Barrett and J. D. Couch. Compiler Construction: Theory and Practice. Chicago, Illinois: Science Research Associates Inc., 1979

[23]. M. Westergaard. CPN Tools 4 Extensions: Part 4: Advanced Communication and Debugging. Available at: https://westergaard.eu/2013/11/cpn-tools-4-extensions-part-4-advanced-communication-and-debugging/, November 2013, Blog entry/, accessed: 20.06.2018

[24]. C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri. OpenFst: A General and Efficient Weighted Finite-State Transducer Library. In Implementation and Application of Automata. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 11–23.

[25]. Graphviz – Graph Visualization Software. Available at: http://www.graphviz.org//, accessed: 20.06.2018

[26]. J. C. Carrasquel. Java/PROSEGA: An extension in CPN Tools for generating languages accepted by FSA and minimized deterministic FSA from a state space. Central University of Venezuela, Caracas, Venezuela, Tech. Rep., October 2015.

[27]. IEEE 802.16 Working Group on Broadband Wireless Access Standards. IEEE Std. 802.16e-2005. Local and Metropolitan Area Network. Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems

[28]. A. V. Morales and M. E. Villapol. Towards Formal Specification of the Service in the IEEE 802.16 MAC Layer for Connection Management. In Proceedings of the 9th WSEAS International Conference on Computational Intelligence, Man-machine Systems and Cybernetics. World Scientific and Engineering Academy and Society (WSEAS), 2010, pp. 140–146

[29]. A. V. Morales and M. E. Villapol. Reviewing the Service Specification of the IEEE 802.16 MAC Layer Connection Management: A Formal Approach. CLEI Electronic Journal, vol. 16, August 2013, pp. 1– 12

Prosega/CPN: расширение CPN Tools для автоматного анализа и верификации систем

[1]*Х. С. Карраскель <julio.carrasquel@yahoo.com>*
[2]*А. Моралес <ana.morales@ciens.ucv.ve>*
[3]*М. Е. Виллаполь <maria.villapol@aut.ac.nz>*
[1]*Римский университет Ла Сапиенца, отдел компьютерной, контрольной и управленческой инженерии, ул. Ариосто 25, г. Рим, 00185, Италия*
[2]*Центральный университет Венесуэлы, Школа компьютерных наук, просп. Пасео Лос-Илюстрес, г. Каракас, 1040, Венесуэла*
[3]*Оклендский технологический университет, Школа инженерии, компьютерных и математических наук, ул. Уэллсли-стрит-восток 55, г. Окленд, 1010, Новая Зеландия*

Аннотация. Верификация и анализ распределенных систем являются чрезвычайно важными задачами, особенно сейчас, когда многие компьютерные системы реализуют критически важные сервисы. Для моделирования и верификации систем полезно сочетать разные методы анализа. В частности, это позволяет применять тот формализм и ту технику анализа, которые лучше подходят для того или иного компонента системы. Комбинация из раскрашенных сетей Петри (CPN, Coloured Petri Nets) и конечных автоматов представляет собой успешную формальную методику моделирования и верификации распределенных систем. В связи с этим в данной статье рассматривается инструмент Prosega/CPN (Protocol Sequence Generator and Analyzer), расширение CPN Tools для поддержки автоматного анализа и верификации. Инструмент реализует несколько функций, таких как генерация минимизированного детерминированного конечного автомата из графа достижимости (occurrence graph) раскрашенной сети Петри, генерация языка и сопоставление конечных автоматов. Это решение поддерживается функцией Simulator Extensions, развитие которой обусловлено необходимостью интеграции раскрашенных сетей Петри с другими формализмами. Инструмент предназначен для поддержки формальной методологии верификации коммуникационных протоколов; однако он может использоваться для верификации других систем, анализ которых включает сравнение моделей на разных уровнях абстракции, например, бизнес-стратегий и бизнес-процессов. В статье приведен подробный пример, в котором инструмент Prosega/CPN используется для анализа части спецификации службы управления соединениями MAC IEEE 802.16.

Ключевые слова: формальные методы; раскрашенные сети Петри; CPN Tools; конечные автоматы; верификация протоколов.

DOI: 10.15514/ISPRAS-2018-30(4)-7

Для цитирования: Карраскель Х. С., Моралес А., Виллаполь М. Е. Prosega/CPN: расширение CPN Tools для автоматного анализа и верификации систем. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 107-128 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(4)-7

Список литературы

[1]. T. Murata. Petri nets: Properties, analysis and applications. Proceedings of the IEEE, vol. 77, no. 4, April 1989, pp. 541–580

[2]. K. Jensen and L. M. Kristensen. Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Berlin, Heidelberg: Springer-Verlag, 2009

[3]. CPN Tools – A tool for editing, simulating, and analyzing Coloured Petri Nets. Доступно по ссылке: http://www.cpntools.org/, дата обращения: 20.06.2018

[4]. M. Westergaard. CPN Tools 4: Multi-formalism and Extensibility. In Application and Theory of Petri Nets and Concurrency. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 400–409

[5]. M. E. Villapol. Modelling and Analysis of the Resource Reservation Protocol Using Coloured Petri Nets. Ph.D. dissertation, University of South Australia, Australia, December 2003

[6]. S. Gordon, L. M. Kristensen, and J. Billington. Verification of a Revised WAP Wireless Transaction Protocol, In Application and Theory of Petri Nets and Concurrency. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 182–202

[7]. B. Han. Formal Specification of the TCP Service and Verification of TCP Connection Management. Ph.D. dissertation, University of South Australia, Australia, April 2004

[8]. J. Billington, G. E. Gallasch, and B. Han. A Coloured Petri Net Approach to Protocol Verification. Berlin, Heidelberg: Springer-Verlag, 2004, pp. 210–290.

[9]. G. Gallasch and L. M. Kristensen. Comms/CPN: A Communication Infrastructure for External Communication with Design/CPN. In Proc. of the Third Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, DAIMI PB-554, pages 75–91

[10]. M. Westergaard and K. B. Lassen. The BRITNeY Suite Animation Tool. In Applications and Theory of Petri Nets and Concurrency. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 431–440

[11]. M. Westergaard. Access/CPN 2.0: A High-Level Interface to Coloured Petri Net Models. In Application and Theory of Petri Nets and Con- currency. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 328–337

[12]. AT&T Researchers – Inventing the Science Behind the Service. Доступно по ссылке: http: //www.research.att.com/evergreen/portfolio/, дата обращения: 20.06.2018

[13]. OpenFST Library. Доступно по ссылке: http://www.openfst.org/twiki/bin/view/FST/WebHome, дата обращения: 20.06.2018

[14]. M. Hulden. Foma: A Finite-state Compiler and Library. In Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: Demonstrations Session. Stroudsburg, PA, USA: Association for Computational Linguistics, 2009, pp. 29–32

[15]. A. Almeida, M. Almeida, J. Alves, N. Moreira, and R. Reis. FAdo and GUItar: Tools for Automata Manipulation and Visualization. In Implementation and Application of Automata. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 65–74

[16]. S. H. Rodger. JFLAP: An Interactive Formal Languages and Automata Package. USA: Jones and Bartlett Publishers, Inc., 2006

[17]. C. Ouyang and J. Billington. Formal Analysis of the Internet Open Trading Protocol. In Applying Formal Methods: Testing, Performance, and M/E-Commerce. Berlin, Heidelberg: Springer-Verlag, 2004, pp. 1–15

[18]. S. Barzegar, M. Davoudpour, M. R. Meybodi, A. Sadeghian, and M. Tirandazian. Traffic Signal Control with Adaptive Fuzzy Coloured Petri Net Based on Learning Automata. In Annual Meeting of the North American Fuzzy Information Processing Society, July 2010, pp. 1–8

[19]. N. Danapaquiame, E. Ilavarasan, N. Kumar, and S. K. Dwivedi. Ratification strategy for web service composition using CPN: A survey. In Proc. of the IEEE International Conference on Computational Intelligence and Computing Research, December 2013, pp. 1–4

[20]. J. Zhu, K. Zhang, and G. Zhang. Verifying Web Services Composition based on LTL and colored Petri Net. In Proc. of the 6th International Conference on Computer Science Education, August 2011, pp. 1127–1130

[21]. ISO/IEC. High-level Petri Nets – Part 1: Concepts, Definitions and Graphical Notation. Software and Systems Engineering, ISO/IEC FDIS 15909-1. Final Draft International.

[22]. W. A. Barrett and J. D. Couch. Compiler Construction: Theory and Practice. Chicago, Illinois: Science Research Associates Inc., 1979

[23]. M. Westergaard. CPN Tools 4 Extensions: Part 4: Advanced Communication and Debugging. Доступно по ссылке: https://westergaard.eu/2013/11/cpn-tools-4-extensions-part-4-advanced-communication-and-debugging/, November 2013, Blog entry/, дата обращения: 20.06.2018

[24]. C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri. OpenFst: A General and Efficient Weighted Finite-State Transducer Library. In Implementation and Application of Automata. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 11–23.

[25]. Graphviz – Graph Visualization Software. Доступно по ссылке: http://www.graphviz.org//, дата обращения: 20.06.2018

[26]. J. C. Carrasquel. Java/PROSEGA: An extension in CPN Tools for generating languages accepted by FSA and minimized deterministic FSA from a state space. Central University of Venezuela, Caracas, Venezuela, Tech. Rep., October 2015.

[27]. IEEE 802.16 Working Group on Broadband Wireless Access Standards. IEEE Std. 802.16e-2005. Local and Metropolitan Area Network. Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems

[28]. A. V. Morales and M. E. Villapol. Towards Formal Specification of the Service in the IEEE 802.16 MAC Layer for Connection Management. In Proceedings of the 9th WSEAS International Conference on Computational Intelligence, Man-machine Systems and Cybernetics. World Scientific and Engineering Academy and Society (WSEAS), 2010, pp. 140–146

[29]. A. V. Morales and M. E. Villapol. Reviewing the Service Specification of the IEEE 802.16 MAC Layer Connection Management: A Formal Approach. CLEI Electronic Journal, vol. 16, August 2013, pp. 1– 12

Simulation-based Verification of System-on-Chip Bus Controllers

[1] *M.M. Chupilko <chupilko@ispras.ru>*
[2] *E.A. Drozdova <drozd_96@mail.ru>*
[1] *Ivannikov Institute for System Programming of RAS,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. The paper presents an approach to verification of commutation components of Systems-on-Chip. The core idea is to verify bus controllers and supporting interface parts connected to a reference model at unit-level. The reference model in the approach is suggested to be written in SystemC so that to be easily adjusted to the required bus parameters. The in-house prototype implementing the approach has been applied to the verification of a Verilog model of Wishbone controller. There is a possibility to extend the approach to support other busses and protocols by development of the interface library.

Keywords: unit-level verification; C++TESK

DOI: 10.15514/ISPRAS-2018-30(4)-8

For citation: Chupilko M.M., Drozdova E.V. Simulation-based Verification of Hardware Bus Controllers. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 4, 2018. pp. 129-138. DOI: 10.15514/ISPRAS-2018-30(4)-8

1. Introduction

This paper is devoted to a problem of a technology of unit-level verification of commutation parts of hardware description level (HDL) models. Each System-on-Chip (SoC) in fact is an HDL model, where IP blocks (intellectual property, proprietary functional units), being parts of the system, are connected according to some traditional communication protocol (Wishbone [1], OCP-IP [2], or something else). To verify it, one has to obtain a golden model to be referred to in the verification process, either to create such a model. In case of IP blocks, their reference models are usually provided by their vendors. In case of the commutation part connecting IP blocks, the situation is more difficult. There might be a standard bus controller with a predefined bus width, or, that is more common, there will be an implementation of the standard protocol. The integration problem also looks quite important, as physical layer of the bus protocol is not the only thing that can be erroneous, but the incompatible logic of data transfers between different IP blocks also might be a weak point.

In this research, we propose a technology of a unit-level verification of communication models, using a SystemC reference model provided by a vendor or created manually. C++TESK [3], C++ library of macros meeting all typical requirements of unit-level verification, including reference modeling, stimulus generation, and coverage estimation, is selected as a basic tool for the technology implementation. This tool supports reference model development at least in two ways: in terms of its macro library and easy attachment to any C++-code.

The rest of this paper is divided into five sections. Section 2 contains more information about communication protocols, including information about Wishbone. Section 3 describes related works in the field of verification. Section 4 presents a proposed approach to unit-level verification. Section 5 studies an example of the approach application. Section 6 discusses the results of the work and outlines directions of future research and development.

2. Communication protocols and Wishbone standard

As such, each communication protocol is a system of rules allowing several entities to transmit data to each other. More specific definition includes descriptions of possible entities and fixes a physical parameters of transferring. Typically, when speaking about communication part of SoC, one means transmission layer between active (e.g., processor blocks) and passive (e.g., RAM) blocks inside of SoC. To implement the standard interface for data transmission between IP-blocks, a bus controller is used. The aim of the standard interface is to decrease the number of integration problems and support possible re-use.

There are different ways to connect IP-blocks together. The simplest one is a point-to-point connection. As an example of point-to-point connection, let us consider OCP-IP [2] (Open Core Protocol International Partnership, see fig. 1), which is a medium layer between blocks and the bus. OCP is oriented to typical master (active component) – slave (passive component) communication. The protocol was proposed several years ago as a first step in development of a single standard and flexible solution in communication.

Another standard of communication in SoC is Wishbone [1]. Being widely distributed, it was selected in this work for being an example for test system development. The Wishbone standard specifies a standard interconnection between computational IP cores. It supports interconnection of few IP cores using such methods as a point-to-point, a shared bus (see fig. 2), a crossbar switch (see fig. 3), and a switched fabric. The first one represents a simple interconnection between two IP cores where the one called Master initiates the data exchange and another one called Slave responds to this call. The second method supports binding of more than two blocks in a consequent order. This method is efficient when the data should be transferred from one IP core to another repeatedly. The third and the fourth methods are similar, also representing the interconnection between several IP cores. In both of them, there is a common bus, connecting more than two Master and Slave cores. The Master core can evoke any Slave connected to this bus but the only one at the cycle. Using the crossbar switch method, several Master-Slave cores can be interconnected simultaneously; using the switch fabric — only one pair of cores is connected.

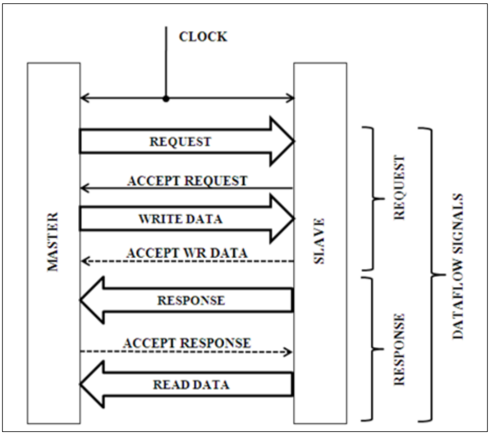


Fig. 1. Open core protocol architecture

From this review, one can derive the following ideas. First, due to the point-to-point connection being the basics of communication in all cases, when testing a bus controller, IP core interfaces should be also taken into consideration. Second, the bus controller is limited in its types of requests (mainly, send and receive); the most important for testing situations seem to be with handling of protocol violations and prevent collisions in bus access.

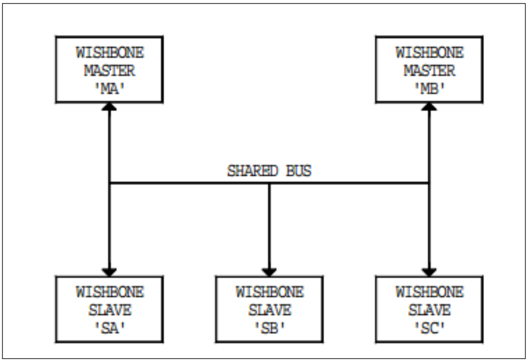


Fig. 2. Wishbone Shared Bus Interconnection

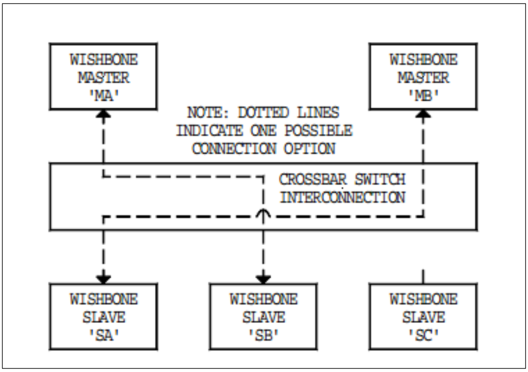


Fig. 3. Wishbone Crossbar Switch Interconnection

3. Related work

The task of SoC verification is typically considered as a problem of a mainly unit-level verification. In this case, a design under test (DUT) is taken separately from its environment. Stimuli are applied and reactions to check are received via DUT wires.

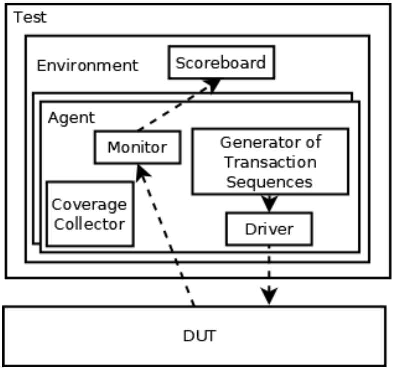


Fig. 4. UVM Test Architecture

Among many unit-level verification approaches, UVM (Universal Verification Methodology) created by Accelera is the most popular. It represents the union of Open Verification Methodology (OVM), Advanced Verification Methodology (AVM) and Universal Reuse Methodology (URM). UVM is a library built upon the SystemVerilog language that provides some basic classes such as the class constructing the testbench structure, the class serving as a basic data structure, the class defining transactions to be passed through components of UVM. This methodology can be used for a constrained random, a coverage-driven, an assertion-based, and emulation-based verification. The UVM testbench structure is as follows (see fig. 4 schematically depicturing UVM test). To begin with, there is the DUT.

The transaction sequencer block serves to interact with the DUT by generating sequences of bits to be transmitted to the DUT. The monitor block is responsible for listening the communication of the DUT and the sequencer, and gets responses from the DUT. The block called scoreboard compares and evaluates all the information that the monitor is receiving from the DUT and the prediction made by the monitor, describing which output is expected to be taken from the DUT. Sequencers, monitors, and coverage collectors together are called agents. An agent and a scoreboard form the environment. At the same time, there is no any explicit method for a making a golden-model as itself, it is up to engineers. The SystemVerilog language is also more a congregation of different methods to describe properties to be checked, rather than a general-purpose programming language convenient for the golden-model development.

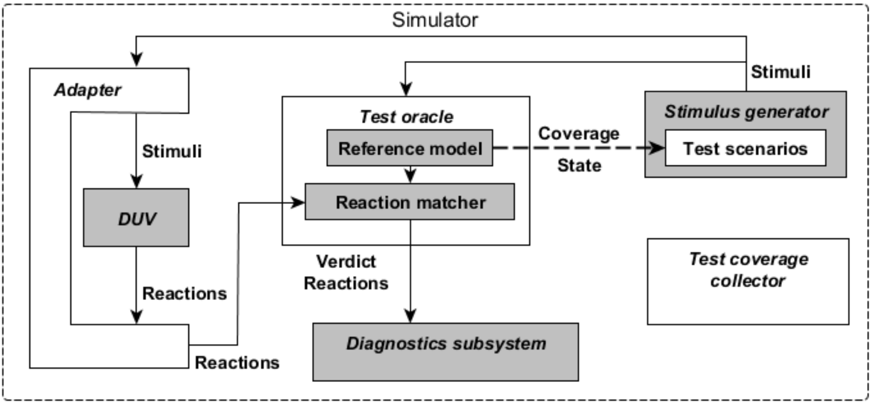


Fig. 5. Verification Environment Architecture

Another approach to unit-level verification is C++TESK Testing Toolkit [3-4], which is represented by a library of C++ macros. The methodology can be used for constrained random and coverage-driven verification. The C++TESK testbench structure (see fig. 5) is similar to the UVM testbench structure but there are some distinctions. According to a selected strategy, C++TESK's stimulus generator chooses one of the predefined stimuli to be sent to the DUT and to the reference model expressed explicitly. The comparator gets the output from the model and from the DUT, compares it and evaluates the coverage. C++TESK is compatible with SystemC, aimed to reference models development, which is one of the key advantages comparatively with UVM.

4. Proposed approach

As it has been already mentioned, either each bus controller is represented by a SystemC model as well as by an RTL description or there is only RTL code and no reference model at all. The conformance between the abstract (SystemC) reference model and the RTL description should be established. Following a simulation-based tradition, it is to be done by verifying both sides against the same specification or by verifying the RTL description against the (SystemC) reference model.

To perform verification, a C++TESK Testing Toolkit has been selected, being a C++ library with all necessary classes and macros for specifying design behavior, generating stimuli, and checking reactions. A C++TESK-based verification environment is structured as shown in fig. 5.

The central component of the verification environment is a test oracle, which is responsible for checking whether the DUT behaves properly. It typically includes a reference model that takes stimuli as an input and produces reference reactions as an output, and a reaction matcher that intercepts reference reactions and implementations reactions provided by the DUT and composes reactions pairs. Usually, the reaction matcher works independently for each output interface.

To transform high-level stimuli to the low-level ones and low-level reactions to the high-level ones, the adapter is used. It consists of multiple interface adapters, each being connected with a single input or output interface. An interface adapter describes a simple protocol of putting a single stimulus or getting a single reaction. A special component of the reaction matcher, called a reaction arbiter, specifies reactions ordering. The task of the reaction arbiter is to choose a reference reaction (if there are any) for a given implementation one. There are two main predefined strategies: (1) model-based arbitration and (2) adaptive arbitration. The first strategy implies that the reference model is an accurate enough to predict reaction order for some output interface (the arbiter selects the next reference reaction stored in the interface buffer). The second strategy is used when the reference model is time inaccurate, in which case, given an implementation reaction, the arbiter searches for a reference reaction being equal or similar to the given one. If some reactions are mismatched, a diagnostics subsystem explains what is wrong with the DUT in terms of incorrect, missing, and unexpected reactions.

Other components of a verification environment are a stimulus generator and a test coverage collector. The stimulus generator creates stimuli by exploring the abstract state space of the reference model. The generator is supplied with a set of available stimuli and a function for abstract state calculation; it tries to apply each stimulus in each reachable abstract state. Speaking about verification of a bus controller, it is natural to consider e.g. the number of messages in the bus channel controllers as being the abstract state (though any other abstraction is possible). Such an adjustable stimulus generator allows to produce hard-to-get-into situations which include those with missing messages in one long packet transmission. The test coverage collector estimates the verification completeness basing on user-defined functional coverage metrics, which is more efficient than simply code coverage.

The typical way of C++TESK usage for unit-level verification in case of its own reference model is described in earlier papers (e.g. [3] or [4]), and the method of connection to SystemC reference model should be developed. To be more precise, SystemC model should include not only the model of DUT itself, but also the model of its environment, including full communication topology. It allows sending complex requests, model collisions, and so on.

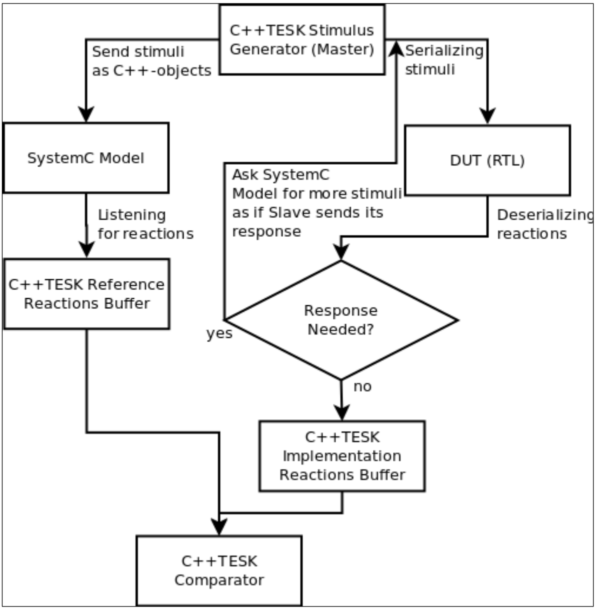


Fig. 6. Architecture of C++TESK, DUT, and SystemC Models Interconnection

The following scheme of commutation between C++TESK and SystemC model is proposed (see Figure 6). C++TESK stimulus generator substitute a master component for the controller bus. Stimuli are applied to both SystemC model (via function calls; to its selected master as if this master wants to send stimuli which the generator applied instead of it) and to DUT (via procedural interface between C and HDL-simulator; to the input master interface of DUT). SystemC computes the behavior of all environment and creates reactions those are to be got from DUT and checked, and those, which are in fact additional stimuli to DUT, e.g. responses from slaves to DUT which are requested by the bus controller and which are substituted by test environment. Output DUT reactions are checked against correspondent SystemC reactions by C++TESK reaction comparator. To provide the program interface between C++TESK and SystemC model, there is a top SystemC class encapsulating all the interfaces between masters and slaves, and the model of bus controller. This class is referenced to in the C++TESK golden model. In order to register reactions from SystemC in proper C++TESK adapters (serializers for stimuli and deserializers for reactions to be checked), there are special listeners of SystemC model activities (more precisely, bool vector of new reactions on different interfaces). When there is some new reaction, it is registered either to be applied to

DUT as a stimulus, or to be added into list of reference reactions to find correspondent implementation reaction in given time.

5. Case study

To develop a prototype of the verification system and to make experiments with it, a Verilog model of simple Wishbone controller has been taken. The controller supports only point-to-point connection. A correspondent SystemC model has been developed from scratch, taking into account the necessity to support different bus sizes and different topologies of master-slave interconnections. The resulted class (see the following listing to see the main part of it) has been attached to C++TESK reference model as an object that should be stimulated by the stimulus generator, and with a possibility of sending reactions that are to be checked against the Verilog model. The whole aspects of the earlier proposed architecture including stimulus generation, HDL implementation reaction checking, and coverage estimation have been kept alive in the prototype of test system. Experiments show that the proposed ideas really work and that future research into this field is required.

```
template <typename adr_bus_size, typename data_bus_size>
  SC_MODULE(Controller) {
    typedef Master<adr_bus_size, data_bus_size> master_type;
    typedef Slave<adr_bus_size, data_bus_size> slave_type;
    // types for containers with masters and slaves
    typedef std::map<master_type*> masters_type;
    typedef std::map<slave_type*> slaves_type;

  ...

  SC_HAS_PROCESS(Controller);
    Controller(sc_module_name _name, bus_mode mode,
               masters_type &masters, slaves_type &slaves):
                 sc_module(name), mode(mode);
    // to register all system's masters and slaves
    // and to bind all masters and slaves to the controller
    void register_master(master_type &master);
    void register_slave(slave_type &slave);
    // to listen for request messages
    void request_listener();
}
```

6. Conclusion

The approach of verification of communication parts of SoC by means of adjustable SystemC reference models and by means of C++TESK's stimulus generator, reaction matcher, and coverage collector has been proposed. Description of the approach includes the architecture of test systems. The idea has been checked in form of a test system prototype for a Verilog model of Wishbone controller. This research should result in the creation of a SystemC library of adjustable models of widely distributed bus standards.

References

[1]. Specification for the WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores. Revision: B.3. Available at: https://cdn.opencores.org/downloads/wbspec_b3.pdf, accessed 20.07.2018

[2]. Open Core Protocol Specification 3.0. Available at: http://www.accellera.org/images/downloads/standards/ocp/OCP_3.0_Specification.zip, accessed 20.07.2018

[3]. M. Chupilko, A. Kamkin. A TLM-Based Approach to Functional Verification of Hardware Components at Different Abstraction Levels. In Proceedings of the Latin American Test Workshop (LATW), 2011, 1-6 pp. DOI: 10.1109/LATW.2011.5985902

[4]. M. Chupilko, A. Kamkin. Runtime Verification Based on Executable Models: On-the-Fly Matching of Timed Traces. In Proceedings of the Model-Based Testing Workshop (MBT), 2013, pp. 67-81. DOI: 10.4204/EPTCS.111.6

Динамическая верификация контроллеров шин систем-на-кристалле

[1] *М.М. Чупилко <chupilko@ispras.ru>*

[2] *Е.А. Дроздова <drozd_96@mail.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. Александра Солженицына, д. 25*

[2] *Московский государственный университет им. М.В. Ломоносова, 119991, Россия, г. Москва, Ленинские горы, д. 1*

Аннотация. В работе представлен подход к верификации коммутационных компонентов систем на кристалле. Основной идеей подхода является верификация контроллеров и поддерживающих интерфейсный обмен частей устройств на модульном уровне с помощью моделей, написанных на SystemC. Эталонные модели в предлагаемой тестовой системе должны быть легко настраиваемыми под требуемые параметры шины. Прототип реализации подхода был применен для верификации Verilog-модели контроллера шины Wishbone. В подходе заложена возможность расширения поддержкой других шин и протоколов посредством разработки библиотеки интерфейсов.

Ключевые слова: модульная верификация, C++TESK

DOI: 10.15514/ISPRAS-2018-30(4)-8

Для цитирования: Чупилко М.М., Дроздова Е.А. Динамическая верификация контроллеров шин систем-на-кристалле. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 129-138 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(4)-8

Список литературы

[1]. Specification for the WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores. Revision: B.3. Доступно по ссылке:https://cdn.opencores.org/downloads/wbspec_b3.pdf, дата обращения: 20.07.2018

[2]. Open Core Protocol Specification 3.0. Доступно по ссылке:: http://www.accellera.org/images/downloads/standards/ocp/OCP_3.0_Specification.zip, дата обращения: 20.07.2018

[3]. M. Chupilko, A. Kamkin. A TLM-Based Approach to Functional Verification of Hardware Components at Different Abstraction Levels. In Proceedings of the Latin American Test Workshop (LATW), 2011, 1-6 pp. DOI: 10.1109/LATW.2011.5985902

[4]. M. Chupilko, A. Kamkin. Runtime Verification Based on Executable Models: On-the-Fly Matching of Timed Traces. In Proceedings of the Model-Based Testing Workshop (MBT), 2013, pp. 67-81. DOI: 10.4204/EPTCS.111.6

К синтезу адаптивных различающих последовательностей для конечных автоматов

[1] *А.С. Твардовский <tvardal@mail.ru>*
[1,2,3] *Н.В. Евтушенко <nyevtush@gmail.com>*
[1] *Национальный исследовательский Томский Государственный университет, 634050, Российская Федерация, г. Томск, пр. Ленина, 36*
[2] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[3] *Национальный исследовательский университет "Высшая школа экономики", 101000, Россия, г. Москва, ул. Мясницкая, д. 20*

Аннотация. Конечные автоматы широко используются при построении проверяющих тестов для управляющих систем с гарантированной полнотой обнаружения неисправностей. В ряде случаев такие тесты достигают экспоненциальной длины относительно размеров автомата-спецификации, что мотивирует исследования по оптимизации проверяющих тестов. Существование последовательностей, различающих каждую пару состояний в автомате-спецификации, может существенно сократить длину теста, если такие последовательности достаточно короткие. Более того, при описании современных систем часто приходится учитывать опциональность неформальной спецификации, и соответственно, использовать методы синтеза тестов для недетерминированных автоматов; последнее в большинстве случаев повышает длину тестов. Адаптивные различающие последовательности существуют чаще, чем безусловные, и, как правило, имеют меньшую длину, что делает их выбор более предпочтительным для синтеза тестов. В настоящей работе мы исследуем свойства адаптивных различающих последовательностей и оптимизируем метод построения таковых для полностью определённых, возможно, недетерминированных конечных автоматов. Предложенный подход основан на ограничении размеров различающего автомата, по которому строится различающий тестовый пример, служащий удобной формой представления адаптивной различающей последовательности. Проведённые эксперименты позволили оценить длину и вероятность существования адаптивных различающих последовательностей для случайно сгенерированных автоматов с различной степенью недетерминизма. Также в работе рассмотрен специальный класс так называемых автоматов без слияний, которые описывают широкий класс реальных систем и обладают «хорошими» для синтеза тестов свойствами; в частности, для таких автоматов практически всегда существуют адаптивные различающие последовательности, если для каждой пары «состояние, входной символ» существует не более трех различных переходов, т.е. степень недетерминизма в автомате не больше трех.

Ключевые слова: конечный автомат; тестовый пример; адаптивная различающая последовательность

DOI: 10.15514/ISPRAS-2018-30(4)-9

Для цитирования: Твардовский А.С., Евтушенко Н.В. К синтезу адаптивных различающих последовательностей для конечных автоматов. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 139-154. DOI: 10.15514/ISPRAS-2018-30(4)-9

1. Введение

Конечные автоматы [1] широко используются при построении тестов для управляющих систем с гарантированной полнотой обнаружения неисправностей. Хорошо известными методами синтеза тестов являются W-метод [2] и его различные модификации для детерминированных и недетерминированных автоматов [3, 4]. При тестировании на основе конечно автоматных моделей проверяющий тест строится относительно заданной модели неисправности, определяющей множество неконформных реализаций, которые могут быть обнаружены по выходным реакциям тестируемой системы на входные последовательности теста. Конечно-автоматные методы синтеза тестов часто основаны на идентификации состояний спецификации, осуществляемой при помощи безусловных или адаптивных различающих последовательностей [5]. Безусловные различающие последовательности фиксируются до начала эксперимента, в то время как следующий входной символ в адаптивной последовательности зависит от реакции тестируемой системы на предыдущие входные воздействия.

Построенные по конечному автомату тесты могут достигать экспоненциальной длины относительно числа состояний исследуемого автомата, что во многом обусловлено высокой длиной различающих последовательностей и мотивирует исследования по оптимизации методов её построения. Известно, что проверяющий тест может быть значительно короче, если спецификация обладает последовательностью, которая различает каждую пару состояний автомата [3, 10], однако, такие последовательности не всегда существуют. Адаптивные различающие последовательности существуют чаще, чем безусловные, и более того, обычно имеют меньшую длину, что делает их более предпочтительным выбором при синтезе тестов.

Для построения адаптивной различающей последовательности часто используется дерево преемников. Однако, для дерева преемников недетерминированного автомата не существует необходимых и достаточных условий существования различающего тестового примера. В [9] авторы предлагают альтернативный подход, основанный на построении различающего автомата, и устанавливают необходимые и достаточные условия существования адаптивной различающей последовательности для такого представления.

В настоящей работе мы рассматриваем проблему построения адаптивных различающих последовательностей для полностью определённых, возможно, недетерминированных автоматов и предлагаем оптимизированный подход к построению таких последовательностей на основе различающего автомата. Эксперименты, проведённые со случайно сгенерированными недетерминированными автоматами, позволили оценить эффективность предложенного подхода и сложность (длину) адаптивных различающих экспериментов.

Структура статьи следующая. Раздел 2 содержит необходимые определения из теории автоматов. Оптимизированная процедура построения адаптивной различающей последовательности с использованием различающего автомата представлена в разделе 3. Раздел 4 содержит результаты проведенных компьютерных экспериментов.

2. Основные определения

В данном разделе мы вводим основные определения и обозначения, взятые преимущественно из работ [6, 9].

2.1 Конечный автомат

Под *конечным автоматом* S понимается пятёрка $(S, I, O, \lambda_{\mathsf{S}}, s_0)$, где S, I, и O – конечные непустые множества состояний, входных и выходных символов соответственно, s_0 – начальное состояние, $\lambda_{\mathsf{S}} \subseteq S \times I \times O \times S$ – отношение переходов. Автомат S называется *недетерминированным*, если для некоторой пары $(s, i) \in S \times I$, существует несколько различных пар $(o, s') \in O \times S$, таких что $(s, i, o, s') \in \lambda_{\mathsf{S}}$, иначе, автомат называется *детерминированным*. Автомат S называется *полностью определённым*, если для каждой пары $(s, i) \in S \times I$ существует переход $(s, i, o, s') \in \lambda_{\mathsf{S}}$; иначе, автомат называется *частичным*. Автомат S называется *наблюдаемым*, если для каждой пары (s, i, o, s_1), $(s, i, o, s_2) \in \lambda_{\mathsf{S}}$ справедливо $s_1 = s_2$.

В настоящей работе мы рассматриваем полностью определённые, возможно, недетерминированные наблюдаемые конечные автоматы. Исключением являются тестовые примеры, которые по определению, являются частичными конечными автоматами и будут рассмотрены далее. Пример полностью определённого недетерминированного конечного автомата представлен на рисунке 1. Данный автомат имеет 4 состояния, 3 входных символа и 2 выходных символа.

Для входного символа i и выходного символа o автомата S, т.е. для входо-выходной пары i/o, состояние s' называется *io-преемником* состояния s, если в автомате существует переход (s, i, o, s'). В наблюдаемом автомате, *io*-преемник всегда определяется единственным образом. Отметим, что *io*-преемник для некоторого состояния s может быть пустым, если в автомате не существует ни одного перехода вида (s, i, o, s'), где s' – произвольное состояние автомата. *Входо-выходной последовательностью* автомата S в состоянии *s* называется последовательность входо-выходных пар $i_1/o_1, \ldots, i_n/o_n$, где *выходная последовательность* $o_1, \ldots, o_n$ представляет собой реакцию автомата на *входную последовательность* $i_1, \ldots, i_n$ в состоянии *s*. Например, для автомата S (рисунок 1) в состоянии 3, входной последовательности 1, 0, 2 соответствует выходная реакция 0, 0, 0. Таким образом, 1/0, 0/0, 2/0 – входо-выходная последовательность в состоянии *s* автомата S.

Конечный автомат S называется *автоматом без слияний* [7], если для каждой пары переходов (s_1, i, o, s_1'), $(s_2, i, o, s_2') \in \lambda_S$, $i \in I$, $o \in O$, имеет место $s_1' \neq s_2'$; другими словами, в автомате без слияний переходы из двух различных состояний по одной входо-выходной паре не могут переводить автомат в одно и то же состояние. Ситуация, когда не пустые *io*-преемники для некоторой пары различных состояний совпадают, называется *слиянием* таких переходов. Например, для автомата S (рис. 1) слияние существует для переходов (2, 2, 1, 3) и (3, 2, 1, 3).

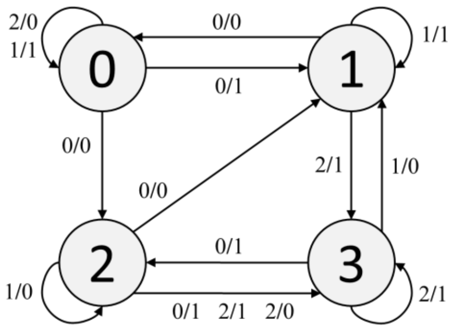


Рис. 1. Полностью определённый недетерминированный автомат S
Fig. 1. Complete nondeterministic FSM S

2.2 Тестовый пример

Входная последовательность α называется *адаптивной*, если следующий входной символ в последовательности зависит от реакции автомата на предыдущий входной символ. Адаптивные входные последовательности часто представляются в форме специального конечного автомата, называемого тестовым примером [8].

Тестовым примером TC(*I*, *O*) с входным алфавитом *I* и выходным алфавитом *O* называется инициальный связный наблюдаемый конечный автомат $T = (T, I, O, \lambda_T, t_0)$ с ациклическим графом переходов. В каждом состоянии тестового примера определены переходы не более чем по одному входному символу, а состояния, в которых не определено ни одного перехода, называются *тупиковыми* (*deadlock*). Тестовый пример TC(*I*, *O*) является частичным автоматом, если $|I| > 1$, и представляет собой адаптивную входную последовательность для автомата $\mathsf{S} = (S, I, O, \lambda_{\mathsf{S}}, s_0)$. *Высота* тестового примера определяется как длина максимальной входо-выходной последовательности из начального в тупиковое состояние и соответствует максимальной длине входной последовательности, которая может быть подана на систему в ходе адаптивного эксперимента.

Тестовый пример TC представляет *адаптивную различающую последовательность* для состояний s_1 и s_2 автомата S, если каждая входо-выходная последовательность из начального в терминальное состояние тестового примера возможна только в одном из состояний s_1 или s_2. В таком случае, TC называется *различающим тестовым примером*, а состояния s_1 и s_2 *адаптивно различимыми*. Адаптивная последовательность, различающая каждую пару состояний автомата S, называется *адаптивной различающей* последовательностью для автомата S. Различающий тестовый пример для автомата на рисунке 1 представлен на рис. 2.

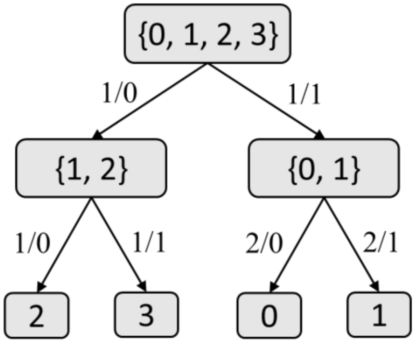


Рис. 2. Различающий тестовый пример для автомата S
Fig. 2. Distinguishing test case for FSM S

Для того чтобы идентифицировать начальное состояние автомата S, различающий тестовый пример TC подаётся на автомат S следующим образом. Эксперимент начинается с подачи на автомат единственного входного символа i_1, определённого в начальном состоянии s_0 автомата TC. Далее различающий пример выполняет переход в состояние s_1 по входо-выходной паре i_1/o_1, где o_1 – наблюдаемая реакция автомата S на входной символ i_1. Далее на автомат S подаётся входной символ, определённый в состоянии s_1. Эксперимент продолжается итеративно, пока в различающем тестовом примере не будет достигнуто тупиковое состояние; по соответствующей входо-выходной последовательности можно однозначно определить состояние, в котором находился автомат S до начала эксперимента.

Например, для различающего тестового примера на рисунке 2 первым будет подан входной символ 1. Далее, по реакции 0 или 1 автомата S (рисунок 1) определяется следующий входной символ 1 или 2 соответственно. Если в ходе эксперимента была получена входо-выходная последовательности 1/1, 2/1, то до начала эксперимента автомат S находился в состоянии 1.

3. Построение различающего тестового примера

Метод построения различающего тестового примера на основе различающего автомата был предложен в [9]. Различающий автомат S_{dist} для полностью определённого наблюдаемого, возможно, недетерминированного автомата $S = (S, I, O, \lambda_S)$ строится по следующим правилам. Множества входных и выходных символов S_{dist} совпадает с таковыми для исходного конечного автомата S, в то время как состояния S_{dist} соответствуют непустым подмножествам состояний S, содержащим более одного элемента; кроме того, в различающем автомате присутствует специальное состояние *F*. Построение переходов S_{dist} начинается в начальном состоянии S_{dist}, которое соответствует подмножеству, содержащему все состояния автомата S. Различающий автомат S_{dist} представляет собой минимальный автомат, построенный по следующим правилам.

Пусть *b* состояние различающего автомата S_{dist}.

1) Переход (*b*, *i*, *o*, *b′*) существует в автомате S_{dist}, если и только если *b′* не является одноэлементным подмножеством, для каждого $o' \in O$ непустые *io′-преемники* для любой пары различных состояний из *b* не совпадают, и *b′* – непустое множество *io*-преемников состояний из множества *b*. В таком случае, состояние *b′* добавляется в множество состояний S_{dist}.
2) Переход (*b*, *i*, *o*, *F*) существует в автомате S_{dist}, если и только если существует выходной символ $o' \in O$, такой что непустые *io′*-преемники для некоторой пары различных состояний из *b* совпадают (имеет место слияние переходов). В таком случае, состояние *F* добавляется в множество состояний S_{dist}.
3) Переход автомата S_{dist} из состояния *b* по входному символу *i* не определён, если и только если для каждого $o' \in O$, не пустые *io′*-преемники любой пары различных состояний из *b* не совпадают, и каждый *io*-преемник состояний из *b* является одноэлементным.

В состоянии *F* существует переход (*F*, *i*, *o F*) для каждой входо-выходной пары *i*/*o* автомата S_{dist}.

Входной символ *i* различающего автомата S_{dist} называется *неопределённым* в состоянии *b*, если в автомате не существует переходов из состояния *b* по входному символу *i*, т.е. для каждого $o' \in O$, не пустые *io′*-преемники любой пары различных состояний из *b* не совпадают, и каждый *io*-преемник состояний из *b* является одноэлементным подмножеством. Для конечного автомата S на рис. 1 соответствующий различающий автомат S_{dist}, представленный на рис. 3, имеет 10 состояний.

Для того чтобы определить, обладает ли конечный автомат S различающим тестовым примером, состояния различающего автомата S_{dist} с неопределённым входным символом удаляются из различающего автомата со всеми переходами в такие состояния. Состояния удаляются итеративно, пока либо в начальном состоянии не появляется неопределённый входной символ, либо в различающем автомате не заканчиваются неопределённые входные символы. При удалении состояния b из S_{dist} мы сохраняем один из входных символов i, неопределённых в состоянии b, в специальном массиве UN, $UN(b) = i$.

Если на некотором этапе удаления состояний, в начальном состоянии различающего автомата существует неопределённый входной символ, то различающий тестовый пример D может быть построен при помощи сохранённых в массиве UN входных символов. В этом случае начальное состояние d_0 автомата D соответствует начальному состоянию b_0 автомата S_{dist}. Входной символ $UN(b_0)$ представляет собой единственный входной символ, определённый в состоянии d_0 автомата D, и переход (d_0, $UN(b_0)$, o, d) существует в D, если и только если в S_{dist} (до удаления состояний) существует переход (b_0, $UN(b_0)$, o, b), где для состояний b и d, соответствующие подмножества состояний автомата S совпадают.

Если для некоторого $o \in O$ не существует перехода (b_0, $UN(b_0)$, o, b) в автомате S_{dist}, то в D добавляется переход (d_0, $UN(b_0)$, o, F), где F – тупиковое состояние. Переходы для других состояний различающего тестового примера D формируются аналогично. Отметим, что для более удобного представления различающего тестового примера, вместо единственного состояния F, можно использовать множество тупиковых состояний, каждое из которых помечается состоянием, в котором находился автомат до начала эксперимента при наблюдении соответствующей входо-выходной последовательности (если таковое существует). Для конечного автомата S на рис. 1 соответствующий различающий тестовый пример представлен на рисунке 2, имеет длину 2 и может быть построен по различающему автомату S_{dist} (рис. 3).

В [9] устанавливается следующее необходимое и достаточное условие существования различающего примера.

Утверждение 1. Для полностью определённого наблюдаемого, возможно, недетерминированного автомата S существует различающий тестовый пример, если и только если в соответствующем различающем автомате S_{dist} не существует полностью определённого подавтомата.

В [9] также показано, что длина различающего тестового примера может достигать значения $2^{n-1} - 1$ для полностью определённого недетерминированного наблюдаемого автомата S с n состояниями. В этом случае, все возможные подмножества состояний автомата S, не являющиеся одноэлементными, могут быть состояниями различающего автомата S_{dist}, и соответственно, процедура построения различающих тестовых примеров становится достаточно сложной.

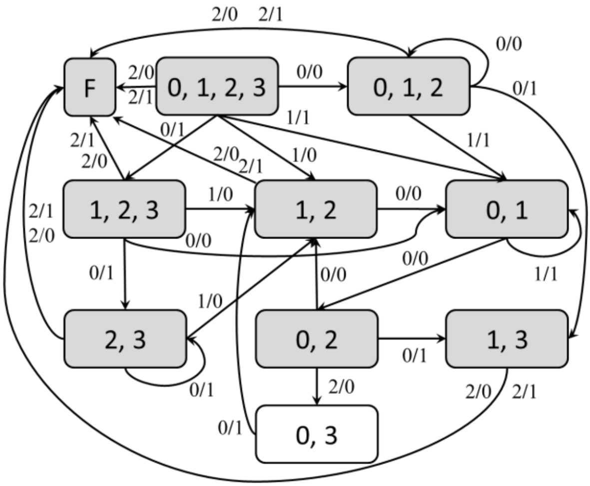


Рис. 3. Различающий автомат S_{dist}
Fig. 3. Distinguishing FSM S_{dist}

Для того чтобы оптимизировать синтез адаптивной различающей последовательности, мы предлагаем ограничивать построение различающего автомата допустимой длиной адаптивной различающей последовательности, а именно, строить различающий автомат S^L_{dist}, содержащий лишь состояния, достижимые из начального состояния по входо-выходным последовательностям длины не больше некоторого $L \geq 1$. Такой сокращённый различающий автомат позволяет найти различающие тестовые примеры высоты не больше L. Предложенный подход согласуется с результатами из [9], где утверждается, что длина адаптивной различающей последовательности не превышает 15 для случайно сгенерированных конечных автоматов даже с большим (более 100) числом состояний. Ввиду того, что длина различающего тестового примера для большинства автоматов не слишком большая, имеет смысл ограничить длину, на которой осуществляется поиск тестового примера.

Таким образом, под автоматом S^L_{dist} понимается различающий автомат, содержащий лишь состояния, достижимые из начального состояния по входо-выходным последовательностям длины не больше L, где L – целое неотрицательное число, и состояние F. Отметим также, что все переходы S^L_{dist} из состояний, достижимых только по входной последовательности длины больше L ведут в состояние F. Например, для автомата на рис. 1, различающий автомат S^0_{dist} содержит только начальное состояние $\{0, 1, 2, 3\}$, в то время как S^1_{dist} содержит еще четыре состояния, достижимых из начального состояния по входной последовательности длины 1 (рис. 3).

Аналогично утверждению 1 из [9], справедлива следующая теорема.

Теорема 2. Для полностью определённого наблюдаемого, возможно, недетерминированного конечного автомата S существует различающий тестовый пример высоты не больше L, если и только если различающий автомат S^L_{dist} не имеет полностью определённого подавтомата.

Доказательство. Пусть в автомате S существует различающий тестовый пример длины L. Тогда по утверждению 1, различающий автомат S_{dist} не имеет полностью определённого подавтомата. Итеративным удалением состояний, достижимых по последовательности длины не больше L можно показать, что для автомата S^L_{dist} также не существует полностью определённого подавтомата. В то же время, если автомат S^L_{dist} не имеет полностью определённого подавтомата, то тестовый пример может быть непосредственно построен по приведённой выше процедуре, аналогичной процедуре из [9].

Таким образом, проверка существования различающего тестового примера высоты не больше L может быть проведена итеративным удалением состояний сокращённого различающего автомата S^L_{dist}. Если начальное состояние S^L_{dist} в результате имеет неопределённый входной символ, то S^L_{dist} не имеет полностью определённого подавтомата и соответствующий различающий тестовый пример может быть построен описанным выше методом.

Например, различающий автомат на рис. 3 имеет 10 состояний, однако, различающий тестовый пример может быть построен на основе автомата S^2_{dist}, который имеет 9 состояний (выделены серым цветом). При удалении состояний с неопределённым входным символом можно убедиться, что различающий автомат S^2_{dist} не имеет полностью определённого подавтомата, и соответствующий различающий тестовый пример (рисунок 2) может быть построен по описанной выше процедуре.

Таким образом, мы предлагаем следующую процедуру построения различающего тестового примера для полностью определённого, возможно недетерминированного, наблюдаемого конечного автомата.

Оптимизированная процедура построения различающего тестового примера

Вход: полностью определённый наблюдаемый автомат S, целое число $L \geq 1$

Выход: Различающий тестовый пример высоты не больше L для автомата S, сообщения ‘Автомат S не имеет различающего тестового примера высоты не больше L’ или ‘Автомат S не имеет различающего тестового примера’

l: = 1, массив неопределённых входных символов $UN := \varnothing$, различающий автомат S^0_{dist} с единственным начальным состоянием, совпадающим с множеством состояний автомата S; Q = S^0_{dist};

Шаг 1. Добавить в S^{l-1}_{dist} состояния, достижимые по входо-выходной последовательности длины l, т.е., построить автомат S^l_{dist} и копировать все новые переходы и состояния в автомат Q.

Если S^{l}_{dist} = S^{l-1}_{dist}, **то** вывести сообщение ‘Автомат S не имеет различающего тестового примера’ и **завершить процедуру**

иначе Шаг 2.

Шаг 2. Итеративно удалить из S^{l}_{dist} каждое состояние *b*, в котором существует неопределённый входной символ, выбрать неопределённый входной символ *i*, положив *UN*(*b*) = *i*.

Если в начальном состоянии S^{l}_{dist} существует неопределённый входной символ, **то** Шаг 3;

иначе

Если *l* + 1 > *L* **то** вывести сообщение ‘Автомат S не имеет различающего тестового примера высоты не больше *L*’ и **завершить процедуру**;

иначе *l* = *l* + 1 и Шаг 1.

Шаг 3. Построить различающий тестовый пример по массиву неопределённых входных символов *UN* и различающему автомату Q, который представляет собой автомат S^{l}_{dist} до удаления состояний.

Завершить процедуру

Теорема 3. Для полностью определённого наблюдаемого, возможно, недетерминированного автомата S и целого числа *L* > 0 предложенная оптимизированная процедура возвращает различающий тестовый пример, если и только если в автомате S существует различающий тестовый пример длины не больше *L*.

Доказательство. Из теоремы 2 и результатов [9] следует, что если автомат S имеет адаптивную различающую длины *L*, то этот различающий тестовый пример может быть построен по различающему автомату S^{L}_{dist}. Покажем далее, что удаление состояний на шаге 2 вышеописанного алгоритма не влияет на построение различающего примера. Пусть для числа *L* был построен автомат S^{L}_{dist} и состояние *b* различающего автомата S^{L}_{dist} имеет неопределённый входной символ. Последнее означает, что для подмножества состояний автомата S, соответствующего состоянию *b*, существует (адаптивная) различающая последовательность длины не более *L* - *L′*, где *L′* – длина входной последовательности из начального состояния S^{L}_{dist} в *b*.

Отметим также, что различающий автомат, соответствующий автомату S^{L}_{dist} до удаления состояний, сохраняется в автомате Q, на основе которого различающий тестовый пример может быть восстановлен по множеству неопределённых входных символов *UN*. Таким образом, состояние *b* будет удалено, как только для него будет найдена кратчайшая (адаптивная) различающая последовательность, которая позже может быть восстановлена по автомату Q и множеству *UN*, и включена в различающий тестовый пример.

Оптимизированная процедура построения различающего тестового примера была реализована программным образом, и далее мы приводим результаты проведённых компьютерных экспериментов.

4. Экспериментальные результаты

В настоящем разделе мы приводим результаты компьютерных экспериментов по оценке эффективности предложенного оптимизированного подхода к построению различающих тестовых примеров, а также исследуем длину адаптивных различающих последовательностей для различных классов конечных автоматов.

4.1 Случайно сгенерированные конечные автоматы

Первая серия экспериментов проводилась со случайно сгенерированными конечными автоматами для оценки вероятности существования адаптивной различающей последовательности в зависимости от числа недетерминированных переходов автомата. Количество nd переходов автомата для каждой пары ‘состояние, входной символ’ задается заранее. Проведённые эксперименты (рис. 4) показали, что различающий тестовый пример существует крайне редко, при $nd > 3$, и соответственно, последующие серии экспериментов проводились с автоматами, для которых $nd \leq 3$.

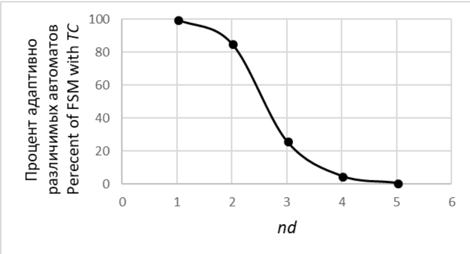


Рис. 4. Процент автоматов, для которых существует различающий тестовый пример
Fig. 4. Percentage of FSMs when a distinguishing test case exists

На рис. 5 представлена средняя высота различающего тестового примера в зависимости от числа состояний автомата. Эксперименты проводились с автоматами, для которых $|I| = |O| = 10$, и которые имели различное число недетерминированных переходов. Нижняя кривая построена для детерминированных автоматов, средняя и верхняя кривая для значений $nd = 2$ и 3 соответственно.

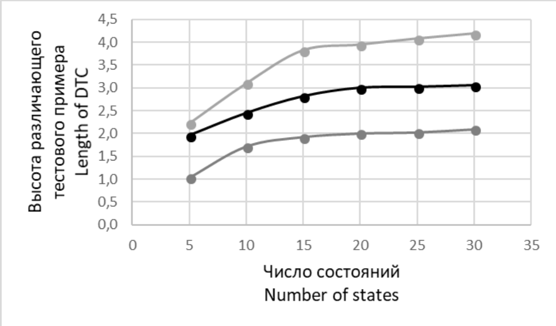


Рис. 5. Высота различающего тестового примера для детерминированных и недетерминированных автоматов
Fig. 5. Length of a distinguishing test case for deterministic and nondeterministic FSMs

Из проведённых экспериментов следует, что для случайно сгенерированных автоматов различающий тестовый пример либо имеет небольшую высоту, либо не существует. Последнее обусловлено тем, что при случайной генерации высока вероятность существования слияний для каждого входного символа. Более того, ввиду большого количества слияний, многие состояния различающего автомата имеют переходы лишь в состояние F, и соответственно, число состояний различающего автомата невелико. Таким образом, предложенная в предыдущем разделе оптимизация оправдана лишь при большом количестве недетерминированных переходов, при котором число состояний различающего автомата и длина различающего примера возрастают.

На рис. 6 отображена зависимость времени построения различающего тестового примера от числа состояний для оптимизированной (нижняя кривая) и неоптимизированной (верхняя кривая) процедур при $|I| = |O| = 10$ и $nd = 3$. Эффективность оптимизированной процедуры возрастает с увеличением числа состояний автомата. Отметим также, что для автоматов с $|S| = 30$, $|I| = |O| = 10$ и $nd = 2$, оптимизированная процедура работает примерно в два раза быстрее оригинальной, в то время как для $nd = 3$ предложенная оптимизация значительно ускоряет выполнение процедуры при прочих равных условиях (рисунок 6).

Заметим, что по результатам проведённых экспериментов случайно сгенерированные автоматы имеют много слияний, что не всегда соответствует автоматам, описывающим поведение реальных систем. В следующем разделе мы рассматриваем специальный класс конечных автоматов без слияний, обладающих рядом полезных свойств при построении различающих тестовых примеров.

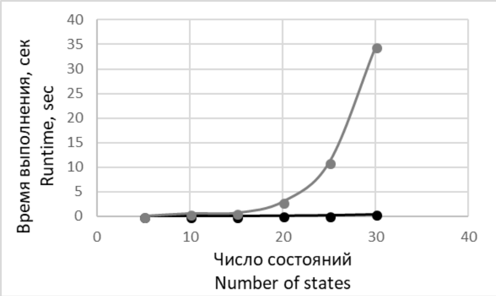


Рис. 6. Время выполнения оригинальной и оптимизированной процедуры построения различающего тестового примера
Fig. 6. The runtime of original and optimized procedures for the distinguishing test case derivation

4.2 Конечные автоматы без слияний

Автомат без слияний обладает адаптивной различающей последовательностью, если и только если каждая пара состояний в нём адаптивно различима, т.е. обладает адаптивной различающей последовательностью [7]. Таким образом, можно предполагать, что для такого класса автоматов тест полиномиальной длины существует чаще. Ввиду последнего, а также того, что автоматы без слияний описывают достаточно широкий класс реальных управляющих систем, такие автоматы представляют особый интерес для исследований.

Зависимость высоты различающих тестовых примеров от числа состояний для случайно сгенерированных автоматов без слияний совпадает с таковой для произвольных автоматов. В частности, для автоматов с $|S| = 100$, $|I| = |O| = 10$ и $nd \leq 2$, высота различающих тестовых примеров не превышает значения 5.

В то же время вероятность существования адаптивной различающей последовательности для автоматов без слияний существенно выше и стремится к 100 процентам для автоматов с $|S| = 100$, $|I| = |O| = 10$ и $nd \leq 2$. Отметим также, что для произвольных автоматов с $|S| = 100$, $|I| = |O| = 10$ и $nd = 2$ время выполнения неоптимизированной и оптимизированной процедур построения различающего тестового примера не превышает 60 секунд, при этом оптимизированная процедура работает примерно в два раза быстрее для автоматов со 100 состояниями. Эффективность оптимизированной процедуры увеличивается с ростом состояний и степенью недетерминизма.

5. Заключение

В настоящей работе была исследована задача построения адаптивных различающих последовательностей для полностью определённых наблюдаемых, возможно, недетерминированных конечных автоматов на основе соответствующего различающего автомата. Предложена оптимизация построения различающего автомата, по которому строится адаптивная различающая последовательность. Было экспериментально показано, что длина адаптивной различающей последовательности для случайно сгенерированных конечных автоматов не достигает «худшей» экспоненциальной оценки.

Однако вероятность существования таких последовательностей уменьшается с увеличением недетерминизма и близка к нулю, когда число переходов для каждой пары «состояние, входной символ» превышает три. Эффективность предложенного подхода к оптимизации была исследована путём проведения компьютерных экспериментов со случайно сгенерированными автоматами. Было также показано, что для специального класса автоматов без слияний, адаптивная различающая последовательность почти всегда существует.

Благодарности

Работа выполнена при поддержке гранта РНФ No. 16-49-03012.

Список литературы

[1]. Гилл А. Введение в теорию конечных автоматов. М., Наука, 1966, 272 стр.
[2]. Chow, T.S. Test design modeled by finite-state machines. IEEE Transactions on Software Engineering, vol. 4, No 3, 1978, pp. 178-187
[3]. Petrenko A. and Yevtushenko N. Conformance Tests as Checking Experiments for Partial Nondeterministic FSM. In Proceedings of the 5th International Workshop on Formal Approaches to Testing of Software (FATES 2005), LNCS 3997, 2005, pp. 118-133
[4]. Dorofeeva R., El-Fakih K., Maag S., Cavalli A.R., Yevtushenko N. FSM-based conformance testing methods: a survey annotated with experimental evaluation. Information and Software Technology, 52, 2010, pp. 1286-1297
[5]. Alur R., Courcoubetis C., Yannakakis M. Distinguishing tests for nondeterministic and probabilistic machines, In Proc. of the 27th ACM Symposium on Theory of Computing, 1995, pp. 363-372.
[6]. Petrenko A., Yevtushenko N. Adaptive testing of deterministic implementations specified by nondeterministic FSMs, In Proc. of the International Conference on Testing Software and Systems, LNCS, vol. 7019, 2011, pp. 162-178
[7]. Yevtushenko N., Kushik N. Decreasing the length of adaptive distinguishing experiments for nondeterministic merging-free finite state machines. In Proceedings of IEEE East-West Design & Test Symposium (EWDTS). 2015. P. 338–341
[8]. Yevtushenko N., El-Fakih K., and Ermakov, A. On-the-fly construction of adaptive checking sequences for testing deterministic implementations of nondeterministic specifications, LNCS, vol. 9976, 2016, pp. 139–152
[9]. El-Fakih K., Yevtushenko N., Kushik N. Adaptive distinguishing test cases of nondeterministic finite state machines: test case derivation and length estimation. Formal Aspects of Computing vol. 30, issue 2, 2018, pp. 319-332
[10]. Tvardovskii A. Refining the Specification FSM When Deriving Test Suites w.r.t. the Reduction Relation. LNCS, vol 10533, 2017, pp. 333-339

[11]. Shabaldina N. Gromov M. FSMTest-1.0: a manual for researches. In Proceedings of the 13th International symposium on IEEE EAST-WEST DESIGN & TEST SYMPOSIUM (EWDTS'15), 2015, pp. 216–219

Deriving adaptive distinguishing sequences for Finite State Machines

[1] *A.S. Tvardovskii <tvardal@mail.ru>*
[1,2,3] *N.V. Yevtushenko < nyevtush@gmail.com>*
[1] *National Research Tomsk State University,*
36 Lenin Ave., Tomsk, 634050, Russia
[2] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[3] *National Research University Higher School of Economics,*
20, Myasnitskaya st., Moscow, 101000 Russia

Abstract. FSM (Finite State Machines) are widely used for deriving tests with guaranteed fault coverage for control systems. Distinguishing sequences (DS) are used in FSM based testing for state identification and can significantly reduce the size of a returned complete test suite. In some cases, length of distinguishing sequence can be exponential with respect to the size of the FSM specification. Moreover, DS can be even longer for non-deterministic FSMs, which are used for the specification optionality description when deriving tests for real systems. Unfortunately, DS not always exist for deterministic and non-deterministic FSMs. Adaptive DS (or corresponding distinguishing test cases (DTC)) are known to exist more often and be much shorter than the preset ones that makes adaptive DS attractive for test derivation. In this paper, we investigate the properties of adaptive DS and propose an approach for optimizing the procedure for the adaptive DS derivation. For this purpose, we propose to limit the height of a DTC and correspondingly to reduce the size of a distinguishing FSM that is used for the DTC derivation in the original procedure. The efficiency of a proposed optimized procedure is evaluated by computer experiments for randomly generated FSMs up to 100 states. We also present the experimental results on checking the percentage of randomly generated FSMs when a DTC exists.

Keywords: Finite State Machine (FSM), test case, adaptive distinguishing sequence.

DOI: 10.15514/ISPRAS-2018-30(4)-9

For citation: Tvardovskii A.S., Yevtushenko N.V. Deriving adaptive distinguishing sequences for Finite State Machines. *Trudy ISP RAN/Proc. ISP RAS*, vol. 30, issue 4, 2018. pp. 139-154 (in Russian). DOI: 10.15514/ISPRAS-2018-30(4)-9

References

[1]. Gill A. Introduction to the Theory of Finite-State Machines. McGraw-Hill, 1964, 207 p.
[2]. Chow, T.S. Test design modeled by finite-state machines. IEEE Transactions on Software Engineering, vol. 4, No 3, 1978, pp. 178-187

[3]. Petrenko A. and Yevtushenko N. Conformance Tests as Checking Experiments for Partial Nondeterministic FSM. In Proceedings of the 5th International Workshop on Formal Approaches to Testing of Software (FATES 2005), LNCS 3997, 2005, pp. 118-133

[4]. Dorofeeva R., El-Fakih K., Maag S., Cavalli A.R., Yevtushenko N. FSM-based conformance testing methods: a survey annotated with experimental evaluation. Information and Software Technology, 52, 2010, pp. 1286-1297

[5]. Alur R., Courcoubetis C., Yannakakis M. Distinguishing tests for nondeterministic and probabilistic machines, In Proc. of the 27th ACM Symposium on Theory of Computing, 1995, pp. 363-372.

[6]. Petrenko A., Yevtushenko N. Adaptive testing of deterministic implementations specified by nondeterministic FSMs, In Proc. of the International Conference on Testing Software and Systems, LNCS, vol. 7019, 2011, pp. 162-178

[7]. Yevtushenko N., Kushik N. Decreasing the length of adaptive distinguishing experiments for nondeterministic merging-free finite state machines. In Proceedings of IEEE East-West Design & Test Symposium (EWDTS). 2015. P. 338–341

[8]. Yevtushenko N., El-Fakih K., and Ermakov, A. On-the-fly construction of adaptive checking sequences for testing deterministic implementations of nondeterministic specifications, LNCS, vol. 9976, 2016, pp. 139–152

[9]. El-Fakih K., Yevtushenko N., Kushik N. Adaptive distinguishing test cases of nondeterministic finite state machines: test case derivation and length estimation. Formal Aspects of Computing vol. 30, issue 2, 2018, pp. 319-332

[10]. Tvardovskii A. Refining the Specification FSM When Deriving Test Suites w.r.t. the Reduction Relation. LNCS, vol 10533, 2017, pp. 333-339

[11]. Shabaldina N. Gromov M. FSMTest-1.0: a manual for researches. In Proceedings of the 13th International symposium on IEEE EAST-WEST DESIGN & TEST SYMPOSIUM (EWDTS'15), 2015, pp. 216–219

Registration protocol security analysis of the electronic voting system based on blinded intermediaries using the Avispa tool

I.A. Pisarev <ilua.pisar@gmail.com>
L.K. Babenko <lkbabenko@sfedu.ru>
Department of Information Security, Southern Federal University,
Taganrog, Rostov region, 347928, Russia

Abstract. Electronic voting systems are a future alternative to traditional methods of voting. It is important to verify the main algorithms on which system security is based. This paper analyzes the security of the cryptographic protocol at the registration stage, which is used in the electronic voting system based on blind intermediaries created by the authors. The registration protocol is described, the messages transmitted between the parties are shown and their content is explained. The Dolev-Yao threat model is used during protocols modeling. The Avispa tool is used for analyzing the security of the selected protocol. The protocol is described in CAS+ and subsequently translated into the HLPSL (High-Level Protocol Specification Language) special language with which Avispa work. The description of the protocol includes roles, data, encryption keys, the order of transmitted messages between parties, parties' knowledge include attacker, the purpose of verification. The verification goals of the cryptographic protocol for resistance to attacks on authentication, secrecy and replay attacks are set. The data that a potential attacker may possess is detected. The security analysis of the registration protocol was made. The analysis showed that the objectives of the audit were put forward. A detailed diagram of the messages transmission and their contents is displayed in the presence of an attacker who performs a MITM-attack (Man in the middle). The effectiveness of protocol protection from the attacker actions is shown.

Keywords: e-voting; cryptographic protocols; cryptographic security; cryptographic protocols security verification

DOI: 10.15514/ISPRAS-2018-30(4)-10

For citation: Pisarev I.A., Babenko L.K. Registration protocol security analysis of the electronic voting system based on blinded intermediaries using the Avispa tool. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 4, 2018, pp. 155-168. DOI: 10.15514/ISPRAS-2018-30(4)-10

1. Introduction

The creation of e-voting systems is a serious problem. There are a number of ready-made systems [1,2] that are used in practice, but they are far from a sufficient level of reliability and the presence of necessary mechanisms, such as complete anonymity of the voter or vote checking opportunity after counting stage. There are also a lot of works, in which perspective methods of conducting electronic voting are considered, based on such principles as homomorphic encryption, including threshold schemes, mix-net, secret sharing schemes and others [3-16]. However, in most cases, the authors of such works show theoretical calculations, from which the basic structural unit of interaction between parties does not follow, namely, cryptographic protocol. Any method on which electronic voting is based, no matter how good it is, loses its security if there are any flaws in the structure of cryptographic protocol that lead to various attacks by the intruder. Thus, the goal of this paper is to test the cryptographic protocol in the important registration stage from various attacks, such as attack on parties' authentication, data privacy and replay-attacks using the Avispa tool [17].

2. Avispa tool

Avispa is a tool for automated security analysis of cryptographic protocols [17]. With the help of Avispa, in the context of the developed protocols, it is possible to verify the parties' authentication, the secrecy of data and protection against replay-attacks. It is impossible to perform integrity checks, in particular, used in protocol CMAC mode (Cipher-based message authentication code) using the Avispa tool. The protocol does not imply the use of timestamps in their classic implementation as a part of message. Instead, the developed system uses a temporary session control by server, in which long live sessions are broke down.

In the paper registration stage is analyzed. Three sides are modeled: user, server-intermediary and main server. The protocol will be analyzed after the phase of common session key distribution between the parties. The protocol will be described in CAS+ [18] language, then translated using the Avispa translator into HLPSL [19]. The check will be carried out using the On-the-Fly Model Checking (OFMC) module, where the verification goals are the transmitted data confidentiality and parties' authentication.

For verification, it is necessary to describe the protocol in one of the formal languages: CAS+ or HLPSL. The first language is simpler in syntax and allows you to quickly describe the protocol. An example of syntax is shown below:

```
protocol NeedhamSchroederPublicKey;
identifiers
A,B                 : user;
Na,Nb               : number;
KPa,KPb  : public_key;

messages
1. A -> B           : {Na, A}KPb
2. B -> A           : {Na, Nb}KPa
3. A -> B           : {Nb}KPb

knowledge
A        : A,B,KPa,KPb;
B        : A,B,KPa,KPb;
```

```
session_instances
 [A:alice,B:bob,KPa:ka,KPb:kb];
```

The second language HLPSL is the language with which Avispa works directly. An example of syntax is shown below:

```
role Alice (A, B: agent,
             KPa, KPb: public_key,
             SND, RCV: channel (dy))
    played_by A def=
    transition

    0.  State  = 0 /\ RCV(start) =|>
        State':= 2 /\ Na' := new() /\ SND({Na'.A}_KPb)
                  ...

 role Bob(A, B: agent,
          KPa, KPb: public_key,
          SND, RCV: channel (dy))
    ...
```

The syntax of this language is more difficult and the best way to describe the protocol is to describe it in CAS+, and then use Avispa to convert it to HLPSL. It is worth to say that if the more complex and larger your protocol, then there is greater chance of errors occurring during translation, so after that you need manually to fix some fragments in HLPSL. It is also worth to say that you should not describe the goals of checking in CAS +, but rather add them directly in HLPSL.

During protocols describing, the following entities are used: roles, data, message order, sessions and verification purposes. After the description of the protocol, including the indication of verification objectives, it is possible to analyze protocol security against attacks. For analyzing, you can use different modes, but the most effective is the OFMC mode (see Fig. 1).

It requires an additional specification for all data involved in the verification, as well as the message area where verification is required for party authentication. As a result of verification, the corresponding result will be issued. In case of attacks detection, the type of attack and its progress will appear in the form of corresponding changes in messages by the intruder, as in Fig. 2.

If there are no attacks, then the program output will contain a corresponding message that protocol is safe (see Fig. 3). Using the «Protocol simulation» button, you can see the interaction scheme of the parties in your protocol. With the help of the button «Intruder simulation» such a scheme will appear, only with the participation of the intruders’ side, in which the data intercepted by him will appear. With the help of the button «Attack simulation» you can see the scheme of the attack with intruder, provided that there is an attack in your protocol.

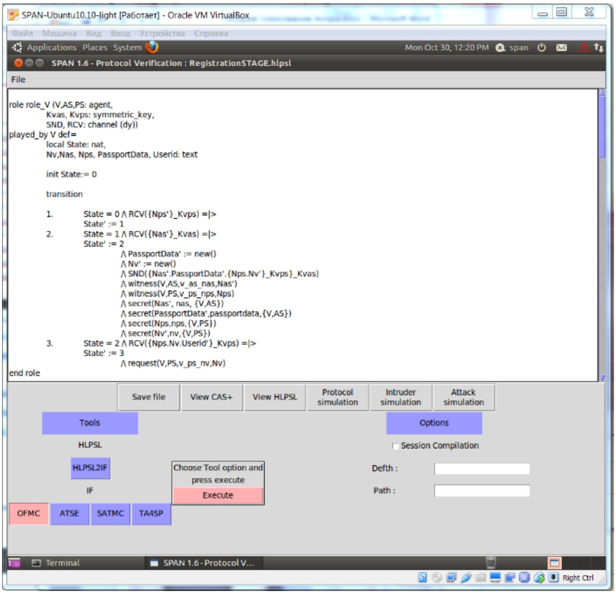


Fig. 1. UI

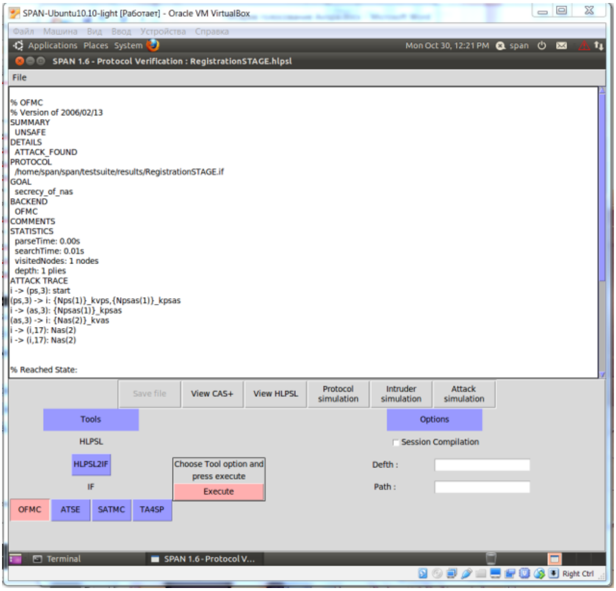


Fig. 2 – Founded attack trace

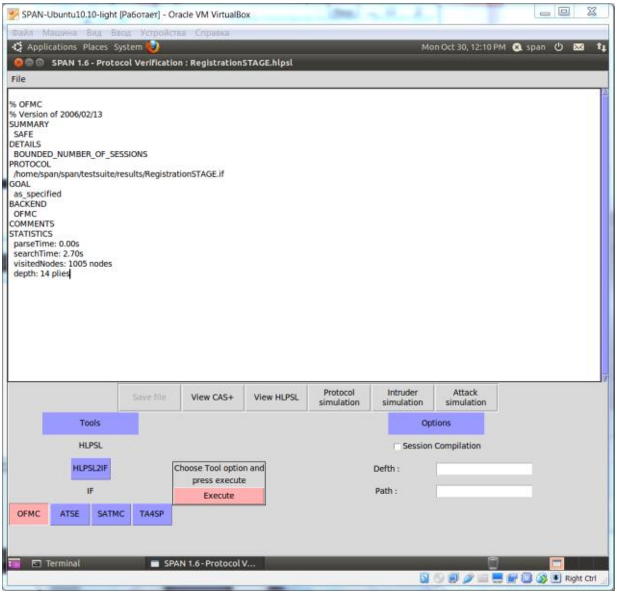


Fig. 3 – Result after verification of safe protocol

2. E-voting system description

2.1 System architecture

The system architecture is based on the use of the following components: client application for voter - V, 3 server applications that will be located on different physical machines: AS (authentication server), PS (processing server), VS (voting server), encryption application for the passport database and ballots DBE (database encryptor). The general scheme of the interaction of components is shown in Fig. 4. The basic principle on which the system protocols are based - blinded intermediaries (see Fig. 5).

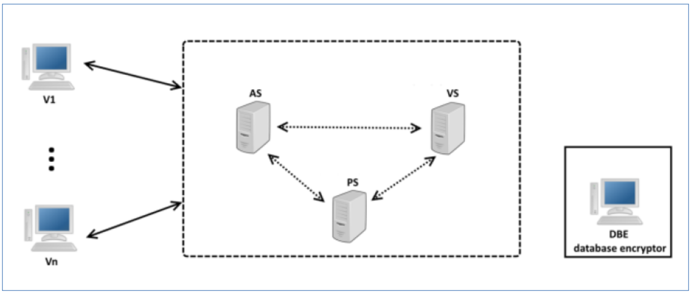


Fig. 4 – System architecture

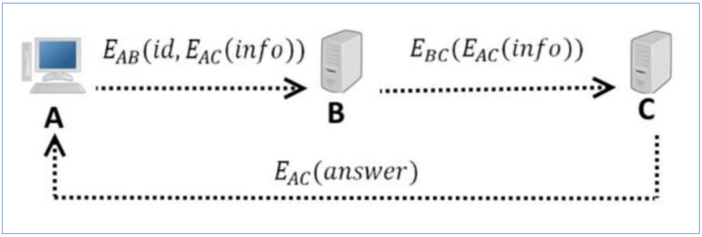


Fig. 5 – Blinded intermediaries principle

There are 3 interacting sides A, B, C. Using the protocol for generating a common secret key, the session key AB, BC, AC are generated. A encrypts some information *info* on the AC key, appends an *id* to it, encrypts it on the AB key and sends this message to B. B in this case is a blinded intermediary, because it can decrypt only the first part of the message with *id*, and the remainder with *info* can not. It accepts the message, decrypts and checks if *id* is in the database and, then redirects the remainder of the message encrypted again on the BC key to the C side. C receives the message, decrypts *info*, encrypts the answer response on the AC key and sends it to A. This principle ensures that: *info* will be accepted only if *id* is in the database and that it is impossible to correlate *id* with *info*.

2.2 Stages description

Stages of electronic voting in the context of the system:

- Preparation. At this stage, a database of voters and a ballot are created. This data is encrypted, and officials deliver this data to the appropriate server components of the system.
- Registration. At this stage, users log in to the system using their identification data, at the moment - using passport data, and they get their anonymous identifier. It should be noted that by using the previously described principle of blind intermediaries, it is impossible to correlate open passport data with an anonymous identifier, which ensures the requirement of anonymity.
- Voting. Users receive a ballot, make their choice and send filled ballot with their anonymous identifier to the server. If such an identifier is present, the vote is accepted, and the verification identifier is sent to user, with which he or she can check vote after counting stage. It is worth noting that it is very important that the user can check his vote after the counting.
- Counting results and votes checking. At the last stage, the votes are counted, the results are published in the public domain, and any voted user can check his or her vote with a verification identifier.

4. Registration stage

The electronic voting system based on blind intermediaries, includes a registration stage in which the voter is given anonymous identifier after presenting his passport data. A simplified scheme of the registration stage is shown in Fig. 6.

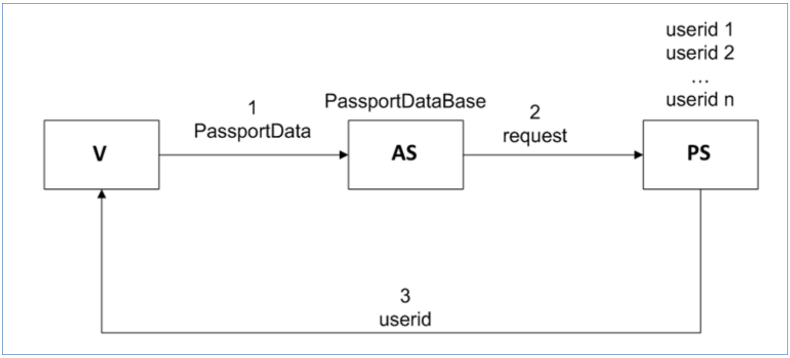


Fig. 6 – Simplified scheme of registration stage.

Secret keys V, VAS, VPS are generated using the protocol for generating a common session key. The server parties generate random numbers and send messages (1), (2), (3) to their recipients. They will be used for parties' authentication. V generates Nv. Next, it generates a message (4) with the passport data, which is a hash from a set of document fields, encrypted random numbers on the shared secret key VPS, calculates the CMAC, encrypts all this data on VAS key, calculates the CMAC and sends to AS. AS in this case is a blinded intermediary. It checks the message integrity by CMAC checking, searches *PassportData* in the database and, if successful, redirects another part of the message (5) to side PS. PS checks integrity, if successful, generates *userid*, adds it to database and sends to V as a message (6). The voter decrypts the message, checks integrity and values of random numbers, and remembers his anonymous unique identifier *userid*, with which the user can vote.

ECDHE (V, AS) - vas
ECDHE (V, PS) – vps
ECDHE (PS, AS) – psas
V: генерирует N_{as}
(1) AS -> V: $E_{vas}(N_{as})$
PS: generates N_{ps}
(2) PS -> V: $E_{vps}(N_{ps})$
PS: generates N_{psas}
(3) PS -> AS: $E_{psas}(N_{psas})$
V: generates N_v.

(4) V -> AS: E_{vas} *(*N_{as}*, PassportData,* E_{vps} *(*N_{ps}*,* N_v*,), CMAC1), CMAC2*

AS -> V: "Success"

(5) AS -> PS: E_{psas} *(*N_{psas}*,* E_{vps}*(*N_{ps}*,* N_v*), CMAC1), CMAC3*

PS: generates userid

(6) PS -> V: E_{vps} *(*N_{ps}*,* N_v*, userid), CMAC4*

ECDHE is a Diffie-Hellman protocol on elliptical curves using ephemeral keys. In our case, we use a modified version of ECDHE-RSA, where authentication is done using a signature RSA and a server certificate which help to prevent MIMT (man in the middle) attacks. The protocol description is as follows.

ECDHE:

(1) V -> S: "Hello"

(2) S -> V: DHs,$Sign_{SKs}$*(DHs),Certificate*

(3) S: Проверяет Certificate и подпись $Sign_{SKs}$*(DHs)*

(4) V -> S: DHv

(5) Both sides generate a common session key K for further interaction with a symmetric cipher.

Here V is the client, S is the trusted server that has the certificate, DHs is the server secret part, DHv is the client secret part $Sign_{SKs}$(DHs) is the signature with the server's private key SKs, Certificate is the server certificate.

When servers generate common secret key, the same protocol is used, except that both parties exchange certificates and if they are valid, a common session key is generated. The security verification of the registration protocol will be carried out after this stage.

5. Security analysis of registration protocol using Avispa tool

Consider the description of the protocol in CAS + at the registration stage.

```
protocol EVotingRegistration;
identifiers
V,AS,PS                                    : user;
Nas,Nps,Npsas,Nv,PassportData,Userid       : number;
Kvas,Kvps,Kpsas                            : symmetric_key;

messages
1. PS -> V    : {Nps}Kvps
2. PS -> AS   : {Npsas}Kpsas
3. AS -> V    : {Nas}Kvas
4. V -> AS    : {Nas,PassportData,{Nps,Nv}Kvps}Kvas
5. AS -> PS   : {Npsas,{Nps,Nv}Kvps}Kpsas
6. PS -> V    : {Nps,Nv,Userid}Kvps

knowledge
```

```
16   V      : V,AS,PS,Nas,Nps,Nv,PassportData,Userid,Kvas,Kvps
17   PS     : V,AS,PS,Nps,Npsas,PassportData,Kvas,Kpsas
18   VS     : V,AS,PS,Npsas,Nps,Nv,Userid,Kvps,Kpsas
19
20   session_instances
21    [V:v,AS:as,VS:ps,Kvas:kvas,Kvps:kvps,Kpsas:kpsas]
22    [V:v,AS:as,VS:ps,Kvas:kvas,Kvps:kvps,Kpsas:kpsas];
23
24   intruder_knowledge
25    v,as,ps;
26
27   goal
28    secrecy_of Nps [V,PS];
29    secrecy_of Npsas [AS,PS];
30    secrecy_of Nas [V,AS];
31    secrecy_of Nv [V,PS];
32    secrecy_of PassportData [V,AS];
33    secrecy_of Userid [V,PS];
34    AS authenticates V on Nas;
35    PS authenticates AS on Npsas;
36    PS authenticates V on Nps;
37    V authenticates PS on Nv;
```

Three interacting parties are described as roles: V, AS, PS (lines 2-3). The identifiers section describes the objects participating in the protocol: interacting parties (line 3), random numbers for authentication, identifiers (line 4). Symmetric keys are specified that will be used for message encryption (line 5). The messages section (lines 7-13) describes the transfer of messages between roles, which data is transmitted, and on which key it encrypted. The knowledge section (lines 15-18) describes roles' data knowledge during the execution of the protocol. In the *session_instances* section (lines 20-22), sessions are described. Among the simulated sessions, 2 are allocated, which allow simulating interaction of two clients with the system. This will detect possible attacks on the parties' authentication and replay-attacks. The *intruder_knowledge* section (lines 24-25) specifies the original knowledge of the intruder. In the goal section (lines 27-37) the secrecy of important values is indicated and the authentication according to the request-response scheme with the transfer of random numbers between the participants. For secrecy of the value, it is necessary that this variable is encrypted and that the encryption key does not come to intruder. In order for one party to authenticate another using the request-response mechanism, it is required that the party wanting to authenticate send a random number to the other party, and that other party in the response message returns this random number. In this protocol there are 4 such actions:

- AS authenticates V by Nas;
- PS authenticates AS by Npsas;
- PS authenticates V by Nps;

- V authenticates the PS to Nv.

As for replay-attacks, protection against them is possible due to the presence of a random number at the beginning of each message, which each side checks when message is received. The results of the check using the OFMC module are shown in Fig. 7. Fig. 8 shows the scheme of interaction between the parties at the stage of registration by steps. Fig. 9 shows the interaction scheme in the presence of an intruder (*Intruder_* side, highlighted in red). This scheme is a visual implementation of the attack man in the middle. When transmitting messages during execution, a transition is made from the «Incoming events» area to «Past events», and the format is the direction of message transfer (from whom and to whom) and the message itself. We can see from the simulation results in the field of intercepted data «Intruder knowledge», all transmitted messages are encrypted on keys which intruder doesn't know, and it excludes the possibility in any way to get important information, such as the user's passport data or unique identifier. The record «nonce-N» means some data that is not readable. Because of the analysis, it was revealed that the registration protocol is safe, ensures the fulfillment of the security objectives (properties) set in the protocol analysis: securing data, authentication of the parties, protection against replay-attacks.

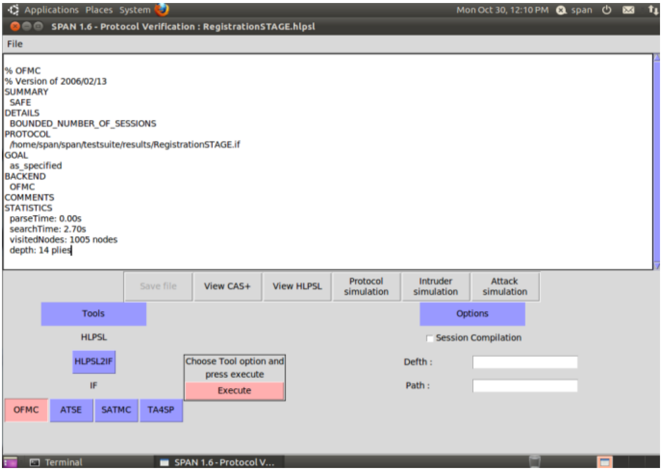


Fig. 7 – Registration protocol verification using OFMC mode.

6. Conclusion

The automated security verification tool Avispa was used for security verification of the registration protocol in electronic voting system based on blind intermediaries, in this paper. The protocol was described in the formal languages CAS+ and HLPSL. The secrecy properties of the transmitted data between the interacting parties were analyzed. It was shown that set security objectives: parties' authentication, verification of data privacy and protection from replay attacks were achieved. The scheme of parties' interaction with the help of tools' graphical functional was considered. An analysis of messages that an intruder can intercept was carried out. Based on the graphical representation it was revealed that all transmitted data is secure, because all messages are encrypted on unknown for intruder keys.

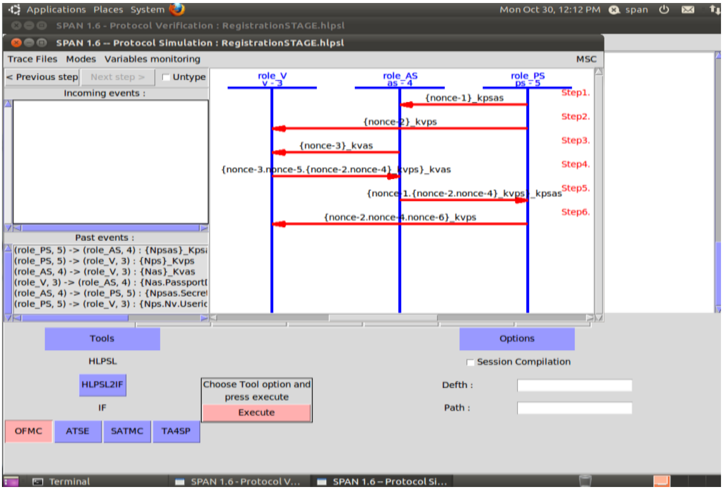


Fig. 8 – Registration protocol in “Protocol simulation” mode.

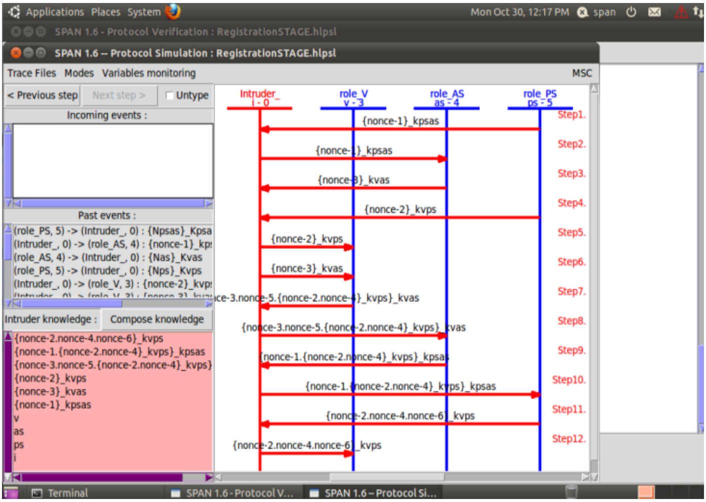


Fig. 9 – Registration protocol verification using OFMC mode.

Acknowledgment

The work was supported by the Ministry of Education and Science of the Russian Federation grant № 2.6264.2017/8.9.

References

[1]. Overview of e-voting systems, NICK Estonia. Estonian National Electoral Commission. Tallinn 2005.
[2]. Dossogne J., Lafitte F. Blinded additively homomorphic encryption schemes for self-tallying voting. Journal of Information Security and Applications, vol. 22, 2015, pp. 40-53.
[3]. Izabachene M. A Homomorphic LWE Based E-voting Scheme. In Proc. of the 7th International Workshop on Post Quantum Cryptography, 2016, pp 245-265.
[4]. Hirt M., Sako K. Efficient receipt-free voting based on homomorphic encryption. In Proc. of the International Conference on the Theory and Applications of Cryptographic Techniques, 2000, pp. 539-556.
[5]. Rivest L. R. et al. Lecture notes 15: Voting, homomorphic encryption. MIT, 2002. Available at http://web.mit.edu/6.857/OldStuff/Fall02/handouts/L15-voting.pdf, accessed 10.06.2018.
[6]. Ben Adida, Mixnets in Electronic Voting, Cambridge University, 2005. Available at http://assets.adida.net/presentations/cambridge-university-voting-2005-01-18.pdf, accessed 10.06.2018.
[7]. Electronic elections: fear of falsification of the results. Kazakhstan today, 2004. Available at http://profit.kz/news/91/Elektronnie-vibori-opasenie-falsifikacii-rezultatov/, accessed 10.06.2018 (in Russian).
[8]. Lipen V.Y., Voronetsky M.A., Lipen D.V., Polevikov E.L. Technology and results of testing electronic voting systems. Ob'edinennyj institut problem informatiki NAN Belarusi [United Institute of Informatics Problems NASB], 2002. Available at http://uiip.bas-net.by/structure/l_kg/results_testing_technology.php/, accessed 10.06.2018 (in Russian).
[9]. David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM, vol. 24, no. 2, 1981, pp. 84-90.
[10]. Ali S. T., Murray J. An Overview of End-to-End Verifiable Voting Systems. arXiv preprint arXiv: 1605.08554, 2016.
[11]. Smart M., Ritter E. True trustworthy elections: remote electronic voting using trusted computing. In Proc. of the International Conference on Autonomic and Trusted Computing, 2011, pp.187-202.
[12]. Bruck S., Jefferson D., Rivest R.L. A modular voting architecture («frog voting»). Towards trustworthy elections. LNCS, volume 6000, 2010, pp. 97-106.
[13]. Jonker H., Mauw S., Pang J. Privacy and verifiability in voting systems: Methods, developments and trends. Computer Science Review, vol. 10, 2013, pp. 1-30.
[14]. Shubhangi S. Shinde, Sonali Shukla, Prof. D. K. Chitre. Secure E-voting Using Homomorphic Technology, International Journal of Emerging Technology and Advanced Engineering, vol. 3, no. 8, 2013, pp. 203-206.
[15]. Neumann S., Volkamer M. Civitas and the real world: problems and solutions from a practical point of view. In Proc. of the Seventh International Conference on Availability, Reliability and Security (ARES), 2012, pp. 180-185.
[16]. Yi X., Okamoto E. Practical remote end-to-end voting scheme. In Proc. of the International Conference on Electronic Government and the Information Systems Perspective, 2011, pp. 386-400.
[17]. The AVISPA team, The High Level Protocol Specification Language. Available at http://www.avispa-project.org/, accessed 10.06.2018
[18]. Ronan Saillard, Thomas Genet, CAS+, March 21, 2011. Available at http://people.irisa.fr/Thomas.Genet/span/CAS_manual.pdf, accessed 10.06.2018
[19]. D. Basin, S. M¨odersheim, and L. Vigan`o. OFMC: A Symbolic Model- Checker for Security Protocols. International Journal of Information Security, vol. 4, issue 3, 2004, pp 181–208.
[20]. L.K. Babenko, I.A. Pisarev, O.B. Makarevich. Secure electronic voting using blinded intermediaries. Isvestiya SFedU. Engineering sciences, no. 5, 2017, pp. 6-15 (in Russian).

Анализ безопасности протокола регистрации в системе электронного голосования на основе слепых посредников с помощью инструмента Avispa

И.А. Писарев <ilua.pisar@gmail.com>
Л.К. Бабенко <lkbabenko@sfedu.ru>
Кафедра информационной безопасности,
Южный Федеральный Университет,
Таганрог, Ростовская область, 347928, Россия

Аннотация. Системы электронного голосования являются будущей альтернативой традиционным способам проведения голосования. Как и для любой системы, важным является верификация ключевых алгоритмов, на которых основана её безопасность. В работе рассматривается анализ безопасности криптографического протокола на этапе регистрации, который используется в созданной авторами системе электронного голосования на основе слепых посредников. Проведено описание протокола регистрации, показаны передаваемые между сторонами сообщения и объяснено их содержимое. При моделировании протоколов предполагается использование модели угроз Долева-Яо. В качестве инструмента для анализа безопасности выбранного протокола используется система Avispa. Протокол описан на языке CAS+ и впоследствии транслирован в специальный язык HLPSL (High−Level Protocol Specification Language), с которым работает используемый инструмент. Описание протокола включает в себя роли, данные, ключи шифрования, порядок передаваемых сообщений между сторонами, знание сторон и злоумышленника, цели проверки. Поставлены цели верификации криптографического протокола на устойчивость к атакам на аутентификацию, секретность и replay-атакам. Установлены данные, которыми может владеть потенциальный злоумышленник. Произведен анализ безопасности протокола регистрации. Анализ показал, что выдвинутые цели проверки были достигнуты. Отображена подробная схема передачи сообщений и их содержимого при наличии злоумышленника, осуществляющего MITM-атаку (Man in the middle). Показана эффективность защиты протокола от действий злоумышленника.

Ключевые слова: электронное голосование; криптографические протоколы; криптографическая защита; верификация безопасности криптографических протоколов

DOI: 10.15514/ISPRAS-2018-30(4)-10

Для цитирования: Писарев И.А., Бабенко Л.К. Анализ безопасности протокола регистрации в системе электронного голосования на основе слепых посредников с помощью инструмента Avispa. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 155-168 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(4)-10

Список литературы

[1]. Overview of e-voting systems, NICK Estonia. Estonian National Electoral Commission. Tallinn 2005.

[2]. Dossogne J., Lafitte F. Blinded additively homomorphic encryption schemes for self-tallying voting. Journal of Information Security and Applications, vol. 22, 2015, pp. 40-53.
[3]. Izabachene M. A Homomorphic LWE Based E-voting Scheme. In Proc. of the 7th International Workshop on Post Quantum Cryptography, 2016, pp 245-265.
[4]. Hirt M., Sako K. Efficient receipt-free voting based on homomorphic encryption. In Proc. of the International Conference on the Theory and Applications of Cryptographic Techniques, 2000, pp. 539-556.
[5]. Rivest L. R. et al. Lecture notes 15: Voting, homomorphic encryption. MIT, 2002. Режим доступа: http://web.mit.edu/6.857/OldStuff/Fall02/handouts/L15-voting.pdf, accessed 10.06.2018.
[6]. Ben Adida, Mixnets in Electronic Voting, Cambridge University, 2005. Режим доступа: http://assets.adida.net/presentations/cambridge-university-voting-2005-01-18.pdf, accessed 10.06.2018.
[7]. Электронные выборы: опасение фальсификации результатов. Казахстан сегодня, 2004. Режим доступа: http://profit.kz/news/91/Elektronnie-vibori-opasenie-falsifikacii-rezultatov/, дата обращения 10.06.2018
[8]. Липень В.Ю., Воронецкий М.А., Липень Д.В., Полевиков Э.Л. Результаты апробирования технологий и систем электронного голосования. Объединенный институт проблем информатики НАН Беларуси, 2002. Режим доступа: http://uiip.bas-net.by/structure/l_kg/results_testing_technology.php/, дата обращения 10.06.2018
[9]. David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM, vol. 24, no. 2, 1981, pp. 84-90.
[10]. Ali S. T., Murray J. An Overview of End-to-End Verifiable Voting Systems. arXiv preprint arXiv: 1605.08554, 2016.
[11]. Smart M., Ritter E. True trustworthy elections: remote electronic voting using trusted computing. In Proc. of the International Conference on Autonomic and Trusted Computing, 2011, pp.187-202.
[12]. Bruck S., Jefferson D., Rivest R.L. A modular voting architecture («frog voting»). Towards trustworthy elections. LNCS, volume 6000, 2010, pp. 97-106.
[13]. Jonker H., Mauw S., Pang J. Privacy and verifiability in voting systems: Methods, developments and trends. Computer Science Review, vol. 10, 2013, pp. 1-30.
[14]. Shubhangi S. Shinde, Sonali Shukla, Prof. D. K. Chitre. Secure E-voting Using Homomorphic Technology, International Journal of Emerging Technology and Advanced Engineering, vol. 3, no. 8, 2013, pp. 203-206.
[15]. Neumann S., Volkamer M. Civitas and the real world: problems and solutions from a practical point of view. In Proc. of the Seventh International Conference on Availability, Reliability and Security (ARES), 2012, pp. 180-185.
[16]. Yi X., Okamoto E. Practical remote end-to-end voting scheme. In Proc. of the International Conference on Electronic Government and the Information Systems Perspective, 2011, pp. 386-400.
[17]. The AVISPA team, The High Level Protocol Specification Language. Available at http://www.avispa-project.org/, accessed 10.06.2018
[18]. Ronan Saillard, Thomas Genet, CAS+, March 21, 2011. Available at http://people.irisa.fr/Thomas.Genet/span/CAS_manual.pdf, accessed 10.06.2018
[19]. D. Basin, S. M¨odersheim, and L. Vigan`o. OFMC: A Symbolic Model- Checker for Security Protocols. International Journal of Information Security, vol. 4, issue 3, 2004, pp 181–208.
[20]. Л.К. Бабенко, И.А. Писарев, О.Б. Макаревич. Защищенное электронное голосование с использованием слепых посредников, Известия ЮФУ. Технические науки, №5, 2017 г., стр. 6-15.

Auto-calibration and synchronization of camera and MEMS-sensors

A.R. Polyakov <polyakov.alx@gmail.com>
A.V. Kornilova <kornilova.anastasiia@gmail.com>
I.A. Kirilenko <y.kirilenko@spbu.ru>
Saint Petersburg State University, Software Engineering
28 Universitetskiy prospect, Petergof, Sankt-Peterburg, Russia, 198504

Abstract. This article describes our ongoing research on auto-calibration and synchronization of camera and MEMS-sensors. The research is applicable on any system that consists of camera and MEMS-sensors, such as gyroscope. The main task of our research is to find such parameters as the focal length of camera and the time offset between sensor timestamps and frame timestamps, which is caused by frame processing and encoding. This auto-calibration makes possible to scale computer vision algorithms (video stabilization, 3D reconstruction, video compression, augmented reality), which use frames and sensor's data, to a wider range of devices equipped with a camera and MEMS-sensors. In addition, auto-calibration allows completely abstracting from the characteristics of a particular device and developing algorithms that work on different platforms (mobile platforms, embedded systems, action cameras) independently of concrete device's characteristics as well. The article describes the general mathematical model needed to implement such a functionality using computer vision techniques and MEMS-sensors readings. The authors present a review and comparison of existing approaches to auto-calibration and propose own improvements for these methods, which increase the quality of previous works and applicable for a general model of video stabilization algorithm with MEMS-sensors.

Keywords: camera calibration; auto-calibration; digital signal processing; rolling shutter; computer vision; grid search

DOI: 10.15514/ISPRAS-2018-30(4)-11

For citation: Polyakov A.R., Kornilova A.V., Kirilenko I.A. Auto-calibration and synchronization of camera and MEMS-sensors. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 4, 2018. pp. 169-182. DOI: 10.15514/ISPRAS-2018-30(4)-11

1. Introduction

The high quality of frames, received from modern smartphone cameras, expands the frontiers of solutions in computer vision tasks. Lately, there are more and more attempts to scale current practices in such areas of computer vision as video stabilization [1], [2], [3], [4], augmented reality[5], 3D reconstruction [6], [7], photogrammetry on mobile platforms and embedded systems. However, these algorithms demand big computational resources that not allows applying them to above-mentioned platforms and in real time.

The presence of numerous different sensors on these platforms, caused by the low cost of their production and high precision at the same time, allows using their data effectively. As the majority of above-stated tasks is any way connected with detection of camera movement (which is the "bottleneck" in most algorithms), the main preference is given to motion sensors – gyroscope and accelerometer [8], [9].

Expansion of mathematical model of computer vision algorithm not only increases quality and reduces calculations but gives rise to new difficulties. In particular, besides general intrinsic parameters of the camera (focal length, optical center, rolling shutter) there are parameters of sensors (i.e, bias for gyroscope) and parameters of model "camera-sensors" (camera and sensors orientation, camera and sensors synchronization parameters). Therefore, if desired to scale an algorithm to a large amount of platforms (for example, in case of mobile phones) automatic calibration of these parameters is needed. It is caused by a big variety of cameras, sensors and their combinations.

This work is a continuation of the research [10] conducted on a subject of real-time digital video stabilization using MEMS-sensors and aims to prototype and implement an algorithm of auto-calibration of key parameters for this task: focal length and parameters of synchronization of frames and gyroscope data.

2. Preliminaries

This section is devoted to basic definitions, general mathematical models, and agreements, which will come out throughout this work.

2.1 Pinhole camera model

Pinhole camera model (fig. 1) is a basic mathematical camera model, which describes a mapping from 3-dimentional real world to its projection onto the image. This mapping satisfies the formula, in which X is coordinates of a point in real world and x is coordinates of its projection. In addition, it depends on camera parameters: f – focal length, (ox, oy) – optical center [11].

$$\begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = \begin{pmatrix} f_x & 0 & -o_x \\ 0 & f_y & -o_y \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix}$$

2.2 Rotation camera model

In case of camera rotation in space using rotation operator R, we get the next relationship between two projections x_1 and x_2 of one point in space X caught at a different time t_1 (rotation R_1) and t_2 (rotation R_2) correspondingly (fig. 2).

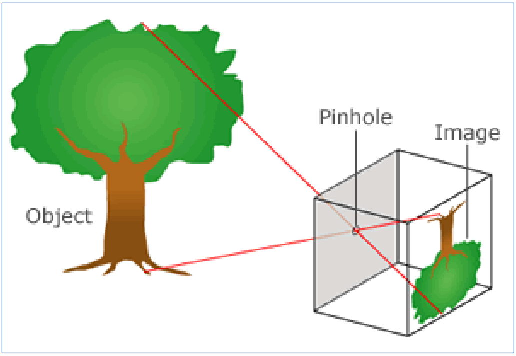


Fig. 1. Pinhole camera model

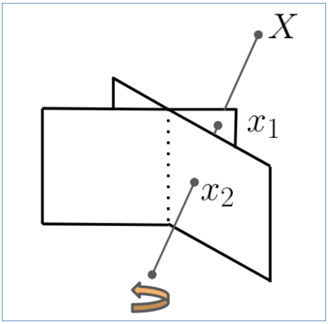


Fig. 2. Rotation camera model

$$x_1 = KR(t_1)X$$
$$x_2 = KR(t_2)X$$

By transforming these expressions, the following needed relationship is established:

$$x_2 = KR(t_2)R^T(t_1)K^{-1}x_1$$

Thus, the matrix of image transformation between moments in time t1 and t2 is defined as:

$$W(t_1, t_2) = KR(t_2)R^T(t_1)K^{-1}$$
$$x_2 = W(t_1, t_2)x_1$$

2.3 Rolling shutter effect

«Rolling shutter» (fig. 3, 4) is an effect arising on the majority of CMOS cameras, at which each row of the frame is shot at different time due to vertical shutter.

When shutter scans the scene vertically, the moment in time at which each point of the frame is shot, directly depends on the row it is located in. Thus, if *i* is the number of the frame and y is the row of that frame, then the moment, at which it was shot can be calculated this way:

$$t(i, y) = t_i + t_s \frac{y}{h},$$

where t_i is the moment when frame number *i* was shot, t_s is the time it takes to shot a single frame, *h* is the height of the frame. This can be used to make the general model more precise, when calculating the image transformation matrix.

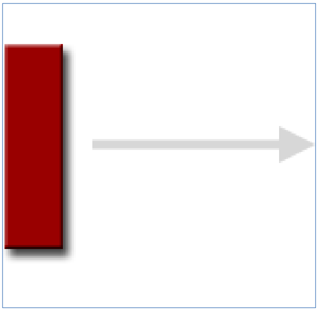


Fig. 3. Object movement

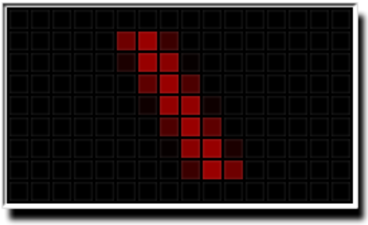


Fig. 4. Rolling-shutter effect during capturing the moving object

2.4 Gyroscope

The gyroscope is a sensor (MEMS-sensor in our case) which sends information about angular velocities of a body. Using this data and its timestamps, a rotation matrix (rotation operator) can be calculated through integration.

There are two approaches for integration data of gyroscope with different computational complexity and accuracy. The first approach is linear integration for receiving Euler angles and then their transformation to a rotation matrix, where θ – is rotation angle of one axis and ω – velocity over this axis between t and t + δ:

$$\theta(t + \delta) = \theta(t) + \int_t^{t+\delta} \omega(t)dt, (2)$$

This approach is applied only in case of insignificant and small rotations, because of the imperfection of Euler angles as an algebraic structure. The other and more complex approach is to use quaternions for data integration. This article [12] gives a full description about the integration of angular velocities using quaternions, and we tend to apply it.

2.5 Stabilization quality metrics

There are two main metrics which can estimate the quality of video stabilization of static scene – RMSE (root mean square error) and ITF (inter-frame transformation fidelity). The first is a comparison between two frames pixel-by-pixel using typical L2 metric. The ITF metric directly depends on PSNR (peak signal-to-noise ratio) parameter between two consecutive frames (k, k+ 1):

$$PSNR(k) = 10 \log_{10} \frac{I_{max}}{MSE(k)},$$

where Imax is maximum pixel intensity, and is counted as:

$$ITF = \frac{1}{N-1} \sum_{k=1}^{N-1} PSNR(k),$$

where N is count of frames in the video.

2.6 Features

In the computer vision, feature is a pattern that satisfies certain properties and can be detected on the image. One of directions of feature use is feature matching, which is mainly focused on searching of similar objects on two frames. In our work, we use feature matching to estimate how the camera moved through shooting.

In our experiments we have used two features types – ORB (Oriented FAST and rotated BRIEF) [13] and SIFT (Scale-Invariant Feature Transform) [14] which prove themselves as the most stable and robust in feature matching. SIFT is considered to exhibit the highest matching accuracies, but requires significant computational resources, while ORB is very fast but less precise [15].

2.7 Description of stabilization algorithm

At the moment stabilization algorithm, proposed in our previous paper [10], works as follows:

1) integrate gyroscope data (angular velocities and timestamps) using quaternions;
2) determine frame timestamp and corresponding rotation matrix;
3) count transformation camera matrix for every horizontal section of the frame (typically, there are several gyro reading per frame and, consequently, several rotation matrices);
4) transform every section using transformation matrix and combine them;
5) write transformed frame to the video.

The algorithm stabilizes video like a tripod, at now complex camera motion is not supported, but in progress.

3. Detailed problem description

As it was mentioned in the description of the stabilization algorithm, it directly depends on camera parameters: focal length, optical center and rolling shutter parameter. In most cases, all parameters besides focal length can be got from API of the device on which this algorithm runs (at the moment the major advantage is given to Android platforms). Thus, one of the main goals of this research is to find focal length, which is the most accurate for our stabilization algorithm.

The other significant direction is to synchronize frames received from the camera and data received from sensors (fig. 5). Mistiming is caused by the time needed for frame processing – scanning and encoding. Therefore, we need to find time offset of this processing to consider it in our model.

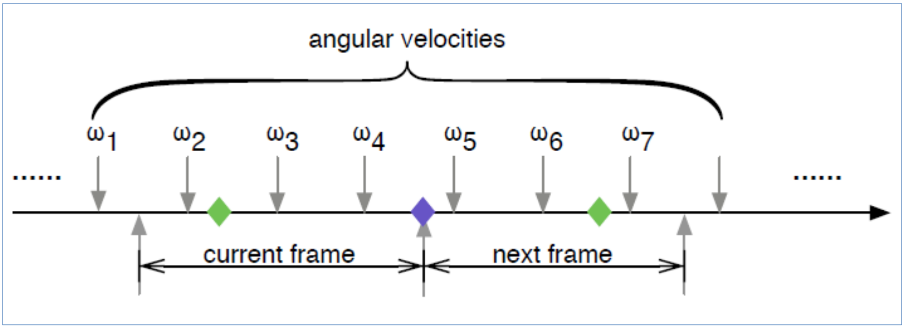


Fig. 5. Matching the time series of frames and gyroscope

Thus, the main goal of this research is to find the suitable focal length and time offset. Some of the described methods are wider and cover other parameters, and we also consider this information.

4. Calibration algorithms

In this section, we describe various approaches that we have tested during this research. The section contains a description of our basic method, review and implementation of the most known methods of calibration from other areas, and our improvements on these methods for our specific task.

4.1 Calibration based on stabilization metrics

focal length, time offset, rolling shutter

This simple approach is based on stabilization metrics described in section 2. Using ITF metric, we can estimate the quality of video stabilization after transformation of frames: the higher the value of metric – the better video is stabilized.

The approach determines three parameters: focal length, time offset and rolling shutter parameter and is as follows: detect a range and step of each parameter (for example, range of focal length – [500, .., 1200] and step – 50) and find tuple of parameters on which metric is maximized using brute-force search.

It is worth noting, despite of the huge computational complexity this method gives the most accurate results due to the strong dependence on the current mathematical model.

4.2 OpenCV calibration method

focal length, optical center, distortion coefficients

This algorithm is applicable only in case of known geometry of subject which is on the scene. Also, the subject should contain easily distinguished feature points. This subject is usually called calibration pattern. We have used use the main calibration pattern which is supported by OpenCV – chessboard. It depends on such parameters as size of chessboard, the distance between cells and others.

The algorithm also determines distortion coefficients and is as follows:

1) count initial intrinsic parameters of the camera. Initial distortion coefficients are equal to zero;

2) estimate camera position using this initial parameters using PnP method;

3) using Levenberg-Marquardt algorithm minimize reprojection error – sum of square root distances between two matched point.

4.3 Grid search method

focal length, time offset

Using frames and gyroscope data, we can estimate the motion of camera in two ways:

1) use feature points on frames and estimate motion using the difference between matched points on consequence frames;

2) use data of gyroscope – measurements and their timestamps.

This approach is as follows. Firstly, we determine two functions which describe the average measure of camera motions in two ways – using feature points and using gyroscope measurements. These functions must depends on time and if necessary must have facilities for interpolation (data of gyroscope is discrete). Having these functions, that describes motion in different ways, we can estimate shift (time offset) of functions using cross-correlation.

Let us determine these functions:

$$r_f(t) = \frac{\sum_{m \in M(t)} (m_x - m'_x) + (m_y - m'_y)}{2|M(t_i)|(t_i - t_{i-1})}$$

$$r_g(t) = \frac{\omega_x(t) + \omega_y(t) + \omega_z(t)}{3}$$

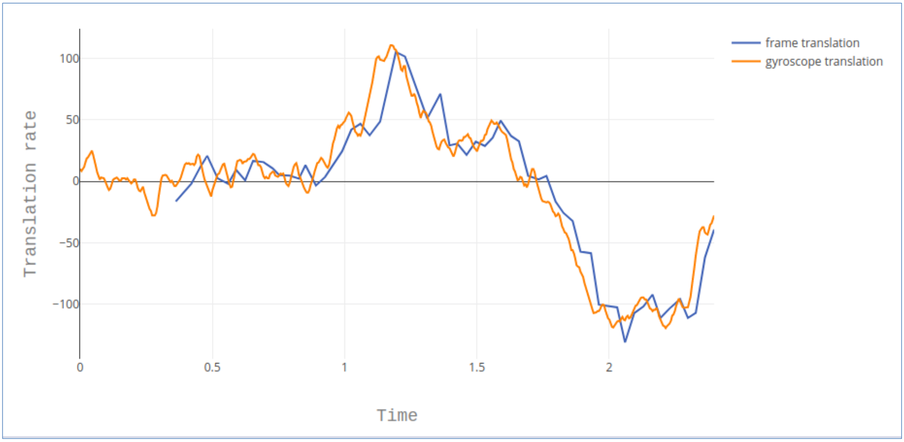


Fig. 6. Time offset between frame and gyroscope

On the picture (fig. 6), you can see similar shape of these functions.

We have tried two typical cross-correlation functions to find offset:

$$s(a,b) = a * b$$

$$s(a,b) = -|a-b|$$

If we have a set of possible offsets Td, we can find offset with a maximum value of correlation between frames and gyroscope functions:

$$offset = \arg\max_{t_d \in T_d} \sum_{t \in T} s(r_f(t-t_d), r_g(t))$$

Authors who support this approach tend to opinion that initial scale constant is a focal length value and try to find this constant like:

$$r_g(t) = f * r_f(t)$$

Using a method of the least squares:

$$f = \arg\min_f \sum_{i=1}^{n} (r_f(t_i + t_d) - f * r_g(t_i))^2$$

4.4 Improvements for grid search method

This method presents a combination of two methods – method with stabilization metrics and method with grid search. The time offset is found by grid search method. If we have a set of possible focal lengths F and the calculated value of time offset, we can calculate a value of focal length. which maximizes stabilization metric:

$$f = \arg\max_{f \in F} ITF(f, t_d)$$

This method is suitable very well in case of using these time offset and focal length in our video stabilization algorithm.

In addition, we have abandoned to take in account motion over zaxis, which is perpendicular to the camera matrix. This motion has non-linear correlation with linear angular velocity over this axis and leads to an error in the algorithm.

5. Results of prototyping

In this section, we will describe results of experiments and conditions in which they were conducted.

5.1 Dataset and environment

Our algorithm was tested on a dataset, which consists of video and gyroscope data from smartphones with the Android operating system. For these purposes, we have a special Android application, which records mp4 video file and csv format file with stamps for gyroscope and frame events. This application supports mobile platforms starting with 21 level Android API because of in this API event-driven scheme for camera frames was supported by camera2 interface. The csv file consists of two types of strings: «f» – for frames and «X, Y, Z, timestamp» – for gyroscope readings.

A framework for calibration algorithm comparison was implemented in Python using OpenCV 3.4 library. It consists of modules for video and gyroscope file parsing and a module for integration of gyroscope readings using quaternion. The framework also has opportunities for calculating metric statics for every method.

We have tested our algorithms on a dataset from the smartphone with the following parameters:

- Model number: Xiaomi Redmi 3S;
- Android version: 6.0.1 (build MMB29M).

5.2 Experiments

Inside our framework, we have implemented all described algorithms and compare them using stabilization quality metrics. We have tested algorithms on different scene types and with different camera movements. An algorithm with stabilization metric was considered as standard. All results are presented in tables. We compare grid search method using different cross-correlation functions and different feature detectors.

Experiments show that OpenCV algorithm has the worst result because of it is very sensitive for the scene (user needs to use chessboard or other pattern) and rotation and is not fit for our mathematical model. In the tables 1-3 you can see results of grid search algorithm without/with improvements (metric) in comparison with stabilization metric algorithm.

The algorithm is parametrized with feature types and shows the best results with the second cross-correlation function (similarity function).

Table 1. Result of calibration in case of 1-dimentional motion

Algorithm	Offset, μs	f	Metric
Metric (standard)	45	850	14.04
Grid Search + ORB	45	825	13.97
Grid Search + SIFT	45	950	13.33
Grid Search + Metric + ORB	45	850	14.04
Grid Search + Metric + SIFT	45	850	14.04

Table 2. Result of calibration in case of 1-dimentional motion

Algorithm	Offset, μs	f	Metric
Metric (standard)	45	850	16.25
Grid Search + ORB	50	850	16.10
Grid Search + SIFT	40	925	15.87
Grid Search + Metric + ORB	50	850	16.10
Grid Search + Metric + SIFT	40	850	15.53

Table 2. Result of calibration in case of 2-dimentional motion

Algorithm	Offset, μs	f	Metric
Metric (standard)	45	850	15.82
Grid Search + ORB	45	950	15.05
Grid Search + SIFT	50	825	15.31
Grid Search + Metric + ORB	45	850	15.82
Grid Search + Metric + SIFT	50	850	15.30

The first two tables show the result of calibration in case of 1-dimentional motion. It is demonstrated that in case of ORB and SIFT features results are identical in accuracy. In addition, results show that in case of metric improvements focal length after calibration is equal to standard in comparison with simple grid search.

The third table describes results of calibration in case of 2-dimentional motion. Results are equal to the case of 1- dimensional motion. As we discussed earlier, the algorithm does not consider 3-dimentional motion because of constraints of grid search model.

5.3 Main results

To sum up, experiments have demonstrated that:

1) grid search method shows the better result for our mathematical model of camera and camera motion;
2) using grid search method, the best calibration result is achieved with the second cross-correlation function (similarity function);
3) ORB and SIFT features show equals results in search of the time offset, therefore we can use ORB as a faster method of feature matching;
4) our improvements of grid search with stabilization metric allow to find focal length which is equal to standard;

5) the algorithm supports only two-dimensional motion (except motion over, axis which is perpendicular to camera matrix), but this is not a strong restriction for users, therefore, our algorithm can be used on a large scale.

6. Conclusion

As lately cameras and motion sensors (gyroscope, accelerometer) very often tend to occur on one platform (smartphones or embedded systems), the quantity of the algorithms, using their joint information, has significantly increased. These algorithms directly depends on parameters of the system «camera-sensors» such as focal length, rolling shutter, synchronization parameters, which differ from platform to platform, and therefore these parameters must be calibrated for increasing of scalability.

Our work proposes the method for auto-calibration of focal length and time series offset (synchronization parameter), which is the most suitable for our video stabilization algorithm using MEMS-sensors. We have review different approaches and choose the nearest for our specific task. We have found parameters for this method, which increase the quality of the calibration algorithm.

It worth noting that proposed algorithm can be scaled not only for stabilization video task. It can be scaled for all algorithms, which support our mathematical model of camera and camera movement.

In the future, we plan to expand the count of calibration parameters with rolling shutter parameter and parameter of relative orientation of the camera and sensor axes.

Acknowledgment

Funding for this work was provided by JetBrains Research.

References

[1]. S. Liu, M. Li, S. Zhu, and B. Zeng. Codingflow: Enable video coding for video stabilization. IEEE Transactions on Image Processing, vol. 26, Issue 7, Apr. 2017, pp. 3291-3302

[2]. M. Grundmann, V. Kwatra, and I. Essa. Auto-Directed Video Stabilization with Robust L1 Optimal Camera Paths. In Proc. of the 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2011), 2011, pp. 225-232

[3]. W.-C. Hu, C.-H. Chen, Y.-J. Su, and T.-H. Chang. Feature-based real-time video stabilization for vehicle video recorder system. Multimedia Tools and Applications, vol. 77, no. 5, Mar. 1, 2018, pp. 5107–5127

[4]. F. Liu, M. Gleicher, J. Wang, H. Jin, and A. Agarwala. Subspace video stabilization. ACM Transactions on Graphics, vol. 30, issue 1, 2011, pp. 4:1-4:10

[5]. D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui. Mobile augmented reality survey: From where we are to where we go. IEEE Access, vol. 5, pp. 6917–6950, 2017

[6]. A. Bethencourt and L. Jaulin. 3d reconstruction using interval methods on the kinect device coupled with an IMU. International Journal of Advanced Robotic Systems, vol. 10, no. 2, 2013

[7]. J. Rambach, A. Pagani, S. Lampe, R. Reiser, M. Pancholi, and D. Stricker. Fusion of unsynchronized optical tracker and inertial sensor in EKF framework for in-car augmented reality delay reduction. In Proc. of the 2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct), Oct. 2017, pp. 109–114

[8]. A. Karpenko, D. Jacobs, and J. Baek. Digital Video Stabilization and Rolling Shutter Correction using Gyroscopes. Stanford Tech Report CTSR 2011-03, 2011, 15 p.

[9]. S. Bell, A. Troccoli, and K. Pulli. A Non-Linear Filter for Gyroscope-Based Video Stabilization. Lecture Notes in Computer Science, vol 8692, 2014, pp. 294-308

[10]. Kornilova A.V., Kirilenko I.A., Zabelina N.I. Real-time digital video stabilization using MEMS-sensors. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 73-86. DOI: 10.15514/ISPRAS-2017-29(4)-5

[11]. R. Szeliski. Computer Vision: Algorithms and Applications. Springer-Verlag London, 2010, 812 p.

[12]. J. Diebel. Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors. Matrix, 58, 2006, pp. 1-35

[13]. E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In Proc. of the 2011 International Conference on Computer Vision, Nov. 2011, pp. 2564–2571

[14]. D. G. Lowe. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, vol. 60, no. 2, pp. 91–110

[15]. E. Karami, S. Prasad, and M. S. Shehata. Image matching using sift, surf, BRIEF and ORB: performance comparison for distorted images. arXiv:1710.02726 [cs.CV], 2017

Автоматическая калибровка и синхронизация камеры и МЭМС-датчиков

А.Р. Поляков<*polyakov.alx@gmail.com*>
А.В. Корнилова <*kornilova.anastasiia@gmail.com*>
Я.А. Кириленко <y.kirilenko@spbu.ru>
Санкт-Петербургский Государственный Университет,
Кафедра Системного программирования
198504, г. Санкт-Петербург, г. Петергоф, Университетский проспект, 28

Аннотация. Данная статья описывает текущие исследования по теме автоматической калибровки и синхронизации камеры и МЭМС-датчиков. Результаты исследования применимы к любой системе, имеющей камеру и МЭМС-датчики, примером которых является гироскоп. Основная задача нашего исследования – нахождение таких параметров системы камера-датчики, как фокусное расстояние камеры и разница во времени между считыванием показания датчика и считыванием кадра камеры, вызванная необходимостью предобработки "сырого" кадра и переводом его в определенный формат. Автоматическая калибровка позволяет применять алгоритмы компьютерного зрения (цифровая видео стабилизация, 3D-реконструкция, сжатие видео, дополненная реальность), использующие кадры видео и показания датчиков, на большем количестве устройств, оснащенными камерой и МЭМС-датчиками. Также автоматическая калибровка позволяет полностью абстрагироваться от характеристик конкретного устройства и разрабатывать алгоритмы, работающие на различных платформах (мобильные платформы, встраиваемые системы, экшн-камеры). Статья описывает общую математическую модель, необходимую для реализации данной функциональности, используя методы компьютерного зрения и показания МЭМС-датчиков. Авторы проводят обзор и сравнение существующих подходов к автоматической калибровке, а также предлагают свои улучшения, повышающие качество существующих алгоритмов.

Ключевые слова: калибровка камеры; автоматическая калибровка; обработка цифровых сигналов; компьютерное зрение

DOI: 10.15514/ISPRAS-2018-30(4)-11

Для цитирования: Поляков А.Р., Корнилова А.В., Кириленко Я.А. Автоматическая калибровка и синхронизация камеры и МЭМС-датчиков. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 169-182 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(4)-11

Список литературы

[1]. S. Liu, M. Li, S. Zhu, and B. Zeng. Codingflow: Enable video coding for video stabilization. IEEE Transactions on Image Processing, vol. 26, Issue 7, Apr. 2017, pp. 3291-3302

[2]. M. Grundmann, V. Kwatra, and I. Essa. Auto-Directed Video Stabilization with Robust L1 Optimal Camera Paths. In Proc. of the 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2011), 2011, pp. 225-232

[3]. W.-C. Hu, C.-H. Chen, Y.-J. Su, and T.-H. Chang. Feature-based real-time video stabilization for vehicle video recorder system. Multimedia Tools and Applications, vol. 77, no. 5, Mar. 1, 2018, pp. 5107–5127

[4]. F. Liu, M. Gleicher, J. Wang, H. Jin, and A. Agarwala. Subspace video stabilization. ACM Transactions on Graphics, vol. 30, issue 1, 2011, pp. 4:1-4:10

[5]. D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui. Mobile augmented reality survey: From where we are to where we go. IEEE Access, vol. 5, pp. 6917–6950, 2017

[6]. A. Bethencourt and L. Jaulin. 3d reconstruction using interval methods on the kinect device coupled with an IMU. International Journal of Advanced Robotic Systems, vol. 10, no. 2, 2013

[7]. J. Rambach, A. Pagani, S. Lampe, R. Reiser, M. Pancholi, and D. Stricker. Fusion of unsynchronized optical tracker and inertial sensor in EKF framework for in-car augmented reality delay reduction. In Proc. of the 2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct), Oct. 2017, pp. 109–114

[8]. A. Karpenko, D. Jacobs, and J. Baek. Digital Video Stabilization and Rolling Shutter Correction using Gyroscopes. Stanford Tech Report CTSR 2011-03, 2011, 15 p.

[9]. S. Bell, A. Troccoli, and K. Pulli. A Non-Linear Filter for Gyroscope-Based Video Stabilization. Lecture Notes in Computer Science, vol 8692, 2014, pp. 294-308

[10]. Kornilova A.V., Kirilenko I.A., Zabelina N.I. Real-time digital video stabilization using MEMS-sensors. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 73-86. DOI: 10.15514/ISPRAS-2017-29(4)-5

[11]. R. Szeliski. Computer Vision: Algorithms and Applications. Springer-Verlag London, 2010, 812 p.

[12]. J. Diebel. Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors. Matrix, 58, 2006, pp. 1-35

[13]. E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In Proc. of the 2011 International Conference on Computer Vision, Nov. 2011, pp. 2564–2571

[14]. D. G. Lowe. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, vol. 60, no. 2, pp. 91–110

[15]. E. Karami, S. Prasad, and M. S. Shehata. Image matching using sift, surf, BRIEF and ORB: performance comparison for distorted images. arXiv:1710.02726 [cs.CV], 2017

Medical Images Segmentation Operations

S.A. Musatian <sabrina.musatian@yandex.ru>
A.V. Lomakin <alexander.lomakin@protonmail.com>
S.Yu. Sartasov <Stanislav.Sartasov@spbu.ru>
L.K. Popyvanov <lev.popyvanov@gmail.com>
I.B. Monakhov <i.monakhov1994@gmail.com>
A.S. Chizhova <Angelina.Chizhova@lanit-tercom.com>
Saint Petersburg State University,
7/9, University Embankment, Saint Petersburg, 199034

Abstract. Extracting various valuable medical information from head MRI and CT series is one of the most important and challenging tasks in the area of medical image analysis. Due to the lack of automation for many of these tasks, they require meticulous preprocessing from the medical experts. Nevertheless, some of these problems may have semi-automatic solutions, but they are still dependent on the person's competence. The main goal of our research project is to create an instrument that maximizes series processing automation degree. Our project consists of two parts: a set of algorithms for medical image processing and tools for its results interpretation. In this paper we present an overview of the best existing approaches in this field, as well the description of our own algorithms developed for similar tissue segmentation problems such as eye bony orbit and brain tumor segmentation based on convolutional neural networks. The investigation of performance of different neural network models for both tasks as well as neural ensembles applied to brain tumor segmentation is presented. We also introduce our software named "MISO Tool" which is created specifically for this type of problems. It allows tissues segmentation using pre-trained neural networks, DICOM pixel data manipulation and 3D reconstruction of segmented areas.

Keywords: deep neural networks; convolutional neural net-works; brain tumors; bony orbit; medical images; segmentation

DOI: 10.15514/ISPRAS-2018-30(4)-12

For citation: Musatian S.A., Lomakin A.V., Sartasov S. Yu., Popyvanov L.K., Monakhov I.B., Chizhova A.S. Medical Images Segmentation Operations. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 4, 2018. pp. 183-194. DOI: 10.15514/ISPRAS-2018-30(4)-12

1. Introduction

Modern ray diagnosis is at the stage of development, and completely different settings and methods are required for different organs: x-ray, MRI, CT, ultrasound are supplemented with invasive contrast methods. Only the doctor can see everything necessary for correct diagnosis and subsequent treatment. However, at the heart of all these methods lie common tasks - the most accurate visualization of the selected zone and obtaining as much data as possible from the results of the examination. In 3D methods (CT and MRI), these tasks are essentially the same, despite the differences in both physical principles and additional settings.

Since the goal of our work is to create a tool that would as accurately as possible visualize isolated structures from raw data obtained by MRI and CT procedures, then this complex work can be decomposed into separate logical components. To isolate complex structures, we formulated the problem of segmentation of tumor processes in MRI images. MRI better visualizes soft tissue and allows to carry out various sequences, change the basic settings of the method in a wide range and use contrast agents. To determine the volume and edge isolation of structures, the problem of determining the volume of bony orbits on a CT was singled out. In this method the bone structures have a high contrast, the distance between slices is very small, and the method itself is widely distributed and takes little time, which allows to study a large data volume.

From the point of medical informatics those problems are not completely dissimilar and could be solved in a unified manner. Moreover, creating a single instrument that may solve all of these challenging tasks autonomously will not only save doctors' time, but also decrease the amount of errors. To the best of our knowledge, there have not been introduced any instrument for automatic segmentation of different body tissues. We came to the conclusion that while the segmentation tasks on different body parts may seem different, they may also all be derived from a core solution based on the deep neural networks.

In this work, we explored state-of-the-art solutions based on deep neural networks for brain tumor segmentation and created an ensemble to see if their performance can be improved and used not only for the brain segmentation task but also for complicated head bony structures in general. We use the results of this research as a first step for creating a convenient and powerful instrument for all medical specialties.

2. Overview

An interest in the possibility of medical images segmentation has increased during the last decade and many different approaches were explored. However, only a few researches evolutionized into complete useful tools for medicine. Commonly used software, that allows semi-automatic segmentation is Brainlab IPlan (commercial distribution) and ITK-SNAP (open source project). The main feature of IPlan, that have already been used in several studies [1, 2], is atlas-based segmentation. Atlas is the described and sketched out by experts shape variations of the ROIs (Region of Interest). Due to complexity of human body structure, there are many problems about the accuracy of delineated atlas. ITK-Snap allows segmentation via active contour evolution method - smooth blow-out of preplaced bubbles into the desired region of interest [3]. Although many of the tasks have been solved by these instruments, there are still many problems that specialists face constantly waiting for improvement. Segmentation is performed by manual or semi-automatic methods.

For the brain tumor segmentation problem many different approaches have been explored and evaluated. There may be formed mainly two classes for these algorithms: methods, which require training on the dataset in advance and those which do not. Early works in this area treated a brain tumor segmentation problem as an anomaly detection problem on the image. Representative works for these approaches may be [4] and [5]. The main advantage of these works is that the presented solutions do not need to be trained beforehand, however that makes it harder to improve the quality of the detection, especially on the smaller tumors.

Another class of approaches is based on the idea of using supervised learning methods, such as random forests [6] or support vector machines [7]. These models can learn a powerful set of features and work quite well on the most common cases, but due to the highly discriminative nature of brain tumors it is hard to detect the correct feature set and create a good model. As a result, recent approaches on segmentation refer to the deep neural networks. It is a powerful instrument that has a capability of extracting new features while training and hence may outperform pre-defined features sets of the supervised learning methods. The results of these algorithms may be also used for different kinds of medical images.

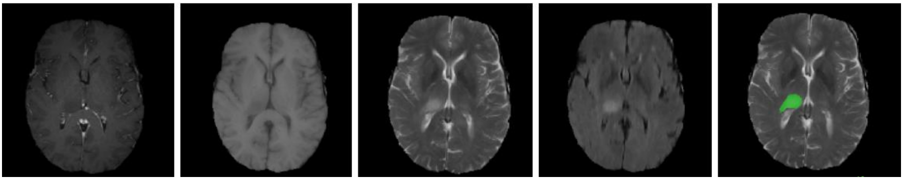
We are developing our own tool - Medical Images Segmentation Operations (MISO), which uses neural networks as a back-end for solving various segmentation tasks in medicine. In the next sections we overview separately application of neural networks for brain tumor and bony orbit segmentation as they were trained and used in MISO.

3. Brain Tumor Segmentation

For that task we chose to overview two CNNs (Convolutional Neural Networks) with different architecture which have proven to be the best in this field: DeepMedic [8] – 11-layers deep, multi-scale, 3D CNN with fully connected conditional random field and WNet [9] – fully convolutional neural network with anisotropic and dilated convolution.

3.1 Data

For the experiments we used BraTS 2017 dataset [10, 11], which includes images from 285 patients of glioblastoma (GBM) and lower grade glioma (LGG). For acquiring this data each patient (fig. 1) was scanned with native T1, post-contrast T1-weighted (T1Gd), T2-weighted (T2), and T2 Fluid Attenuated Inversion Recovery (Flair). For all patients ground-truth segmentation was provided.



(a) *(b)* *(c)* *(d)* *(e)*

Fig.1. Original data from BRaTS 2017 dataset: a) T1Gd b) T1 c) Flair d) T2 e) Ground-truth

3.2 Implementation Details

For WNet we used configuration described in the original papers and BRaTS 2017 dataset for training. For DeepMedic we trained two versions of this network on different datasets and injected some changes into original architecture of this network. For the first version we introduced the following changes: model was trained only on T1 and T2 images.

The reason for that change was that these are the most common MRI sequences. Having a network trained only with this data makes the model available for more hospitals in future. Also, instead of PReLu non-linearity, introduced in the original model, we use SELU [12], which improves the performance and time spend on training. For the second version of DeepMedic we also used SELU, but this network was trained only on T1 images. We wanted to explore how this network will cope when having only one source. For all of these three networks we separated initial dataset into 3 chunks: training (about 80% percent), validation (10%) and test (10%). The performance of these networks on test data may be seen at Table 1. In the observed studies, authors were aiming not only to detect the tumor but also to segment the tumor into three categories: whole tumor, tumor core and enhancing tumor core. However, in our work we are only interested in the whole tumor detection problem.

Table 1. Individual performance of observed CNNs

Network	Dice coefficient
Wnet	0.9148
DeepMedic (inputs: T1+T2)	0.8317
DeepMedic (inputs: T1)	0.6725

3.3 Detecting the Percentage of False Negative Segments

The original works analyse the quality of CNN performance based on the Dice and Hausdorff measurements, which are good for the segmentation problems in general, but hides the necessary details about misclassifications. For that reason, we explored the results from work of the considered networks to determine the percentage of false positives via false negatives results. Our main goal was to examine whether these methods are more prone to predict false positives then false negatives.

Since the decisive opinion during the diagnosis and treatment is always on doctor, our main goal is to indicate if there may be a pathological tissue and get the surgeon's attention to this area. Our system is aiming to find all suspicious areas and send them for reevaluation to medical specialist. Hence, one of the main qualities of this system that should be optimized first-hand would be not false positive results, but false negatives, because those when unnoticed may not get the essential medical care and be a reason for future proliferation of tumor cells. The results of this experiment may be seen at Table 2.

Table 2. Number of false positive via false negative in the final segmentation

Network	mean (False positive / ground truth)	mean (False negative / ground truth)
Wnet	0.0863	0.0830
DeepMedic (inputs: T1+T2)	0.2330	0.1170
DeepMedic (inputs: T1)	0.4690	0.2455

3.4 Neural Network Ensembles

We wanted to detect whether the general performance of these three networks can be improved, when they are used together. So, we proposed the idea of forming the neural networks ensemble [13] out of them. We implemented the following voting scheme: for each voxel we determine each individual result for every neural network, based on their already pre-trained models, and then we qualify a voxel as part of the tumor if and only if the majority of networks classify it as tumor, otherwise it is considered to be a healthy matter. The results of this experiment may be seen at Table 3.

Table 3. The performance of neural network ensemble. The results of combining networks together differently

CNN 1	CNN 2	CNN 3	Dice coefficient
Wnet	DeepMedic (inputs: T1 + T2)	-	0.8861
DeepMedic (inputs: T1+T2)	DeepMedic (inputs: T1)	-	0.7657
DeepMedic (inputs: T1)	Wnet	-	0.7941
DeepMedic (inputs: T1+T2)	DeepMedic (inputs: T1)	Wnet	0.8823
DeepMedic (inputs: T1+T2)	DeepMedic (inputs: T1)	Wnet	0.8823

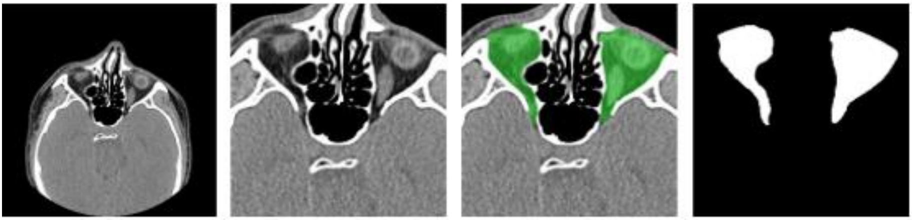
4. Bony Orbit Segmentation

4.1 Methods

Our approach consists of two steps. First of all, image classification was presented, dividing initial dataset into two groups: «contains orbit» and «does not contain orbit». The next step is to segment the orbit in the images marked by the classifier in the previous stage. In this paper first step is described in details, whereas the second step is introduced briefly as it is the subject of further research.

4.2 Data Collection

Raw CT scans was presented by faculty of Medicine of Saint Petersburg State University. Using Toshiba Scanner as instrument and Helical image acquisition as main method, 5 series were made and anonymized. The initial image dimensions were 512*512, using short (2-byte number) to represent radiation intense with Grayscale Standard display function. Orbits occupy less than 1/4 of the image, so we reduced the original size from 512*512 to 256*256 in order to decrease computation complexity (fig. 2 b). Slices with orbit was labeled and some of them was manually segmented by expert (fig. 2 c). Total amount of data: 601 sinus + 80 head CT images were marked as «contains orbit» and 1414 were marked as «doesn't contain orbit». 150 images were segmented.



(a) (b) (c) (d)

Fig. 2. Data for bony orbit segmentation: a) Initial image b) cropped image c) segmented by expert d) extracted mask (label for cropped image)

4.3 Model Choosing

To achieve best classification performance of 1st CNN, some important parameters like number of layers and convolutional kernel size must be chosen. So, several kernel sizes and layers number have been evaluated for classification accuracy. The quantitative assessments are shown in Table 4. As a result, the model used for training consisted of eight layers, out of which four were convolutional layers and four were fully connected layers. The output of last fully-connected layer has been fed to a sigmoid function, as it is a standard neural network classification layer [14]. The initial images were cropped and compressed in order to reduce training time.

Hence, network accepts grayscale images of dimension 128 × 128 as inputs. The first layer filters input with 32 kernels of size 5 * 5.

As it could be seen from experiments, rectified linear unit (ReLU) [15] nonlinearity applied to the outputs of all convolutional layers gives best result compared with other activation functions. The (n+1)th convolutional layer takes the output of nth as input processed by ReLU nonlinearity and max pooling layer respectively and process it with Fn + 1 filters. Filters configuration are shown in Table IV. All fully connected layers have equal number of neurons i.e., 256. For the Second CNN the U-net architecture [16] was chosen, as it has already proven its suitability for segmentation in general. Several layer sequences were evaluated to find most fitting model. In order to reduce bias and increase universality, 2 dropout layers with dropout rate equals to 0.2 were added.

Table 4. Quantitative assessments of different CNN configurations

Neurons in each FCLs*	1st CVL* kernel	Filters model	val.acc.
3200	11	32-64-128-128	0.725
256	11	32-64-128-128	0.9964
3200	7	32-64-128-128	0.7821
512	7	32-64-128-128	0.9782
512	7	64-64-128-256	0.9295
512	11	32-64-128-128	0.9964
256	7	32-64-128-128	0.8214

FCL – fully-connected layers, CVL – convolutional layer, val. acc. – accuracy on validation dataset

4.4 Training Details

Classification CNN was implemented, trained and evaluated using Python 3.6 as programming language on NVIDIA GTX740M GPU with CUDA Toolkit 9.0 and CuDNN 7.0.5. Keras 2.1.*(version was continuously updated during development) was chosen as neural networks framework, working on top of Tensorflow 1.5*. We have trained and evaluated CNNs on a range different filter models (number of filters in each convolutional layer), kernel sizes and neuron amount in fully-connected layers. Also experiments with dropout layer [17] were performed.

4.5 Output Image Visualization

After segmentation has been performed, series of marked images are converted to voxel grid using initial DICOM metadata in order to create 3D model using Marching cubes algorithm by means of MISO Tool and The Visualization Toolkit library. Result is presented in fig. 3.

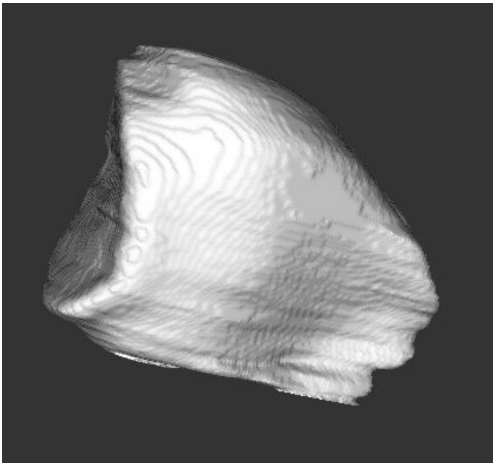


Fig. 3. Rendered bony eye orbit using marching cubes algorithm

4.6 Experimental Results

4.6.1 Images cropping

As the main purpose of our work is to create an instrument, that could be run on our servers from multiple clients, in order to deliver the best performance to the customers and lessen waiting time, computation complexity must be decreased as much as possible. To achieve that goal, it was decided to perform experiments with cropped and resized images. When the image was reduced to less than *128*128*, we were unable to achieve the required accuracy. The best result under the condition "*accuracy > 0.95*" showed the approach in which a piece of *256*256* was cut out of the image, which subsequently was compressed to 128. Because of high similarity of head position in CT scans, it was not necessary to move the cropping window.

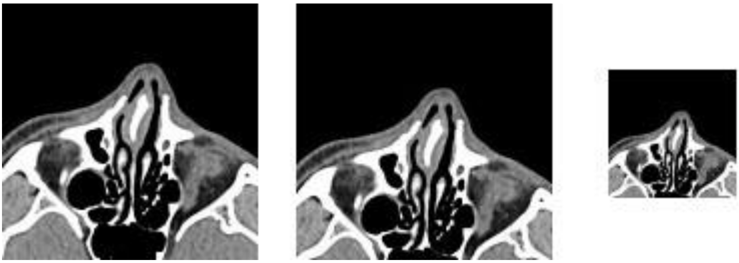


Fig. 4. Different cropping window positions and sizes were examined

4.6.2 Performance

For the 1st CNN we used different kernels from 3 to 11 pixels, different CNN model configurations, activation functions and a suitable epoch number to illustrate which one of these properties support CNN to get the highest level of performance. Data was split between train and validation in proportion 4:1. Our model performs best after 115 training epochs - validation accuracy 99% and then stabilizes. Dropout layers with dropout rate lower than 0.4 doesn't impact the accuracy significantly, and more than 0.4 fails the accuracy to ~85%, so it was decided to exclude dropout layers from final model. Worth noticing the fact that models with 512 neurons in each FCL showed approximately same result as a model with 256 neurons, but it takes up to 1.4 times more computation time, so 256 was chosen as less resource-consuming.

5. Conclusion

In this paper, the first step for the medical segmentation system was introduced. Based on the existing CNN solutions we demonstrated that they may be easily adapted for the segmentation tasks on different medical images. Also, in this work has been shown that these segmentations may be used for creating 3D models and volume estimation. Based on the obtained results, the target tool model was developed using C# 7.0 as programming language and .NET 4.7 as framework.

As the development is still in the very beginning, there is no purpose for service hosting, although it is considered as the only possible option for the further development, so for now MISO (Medical Images Segmentation Operations) tool has been prototyped as a classic desktop application with CNN results visualization abilities (fig. 5)

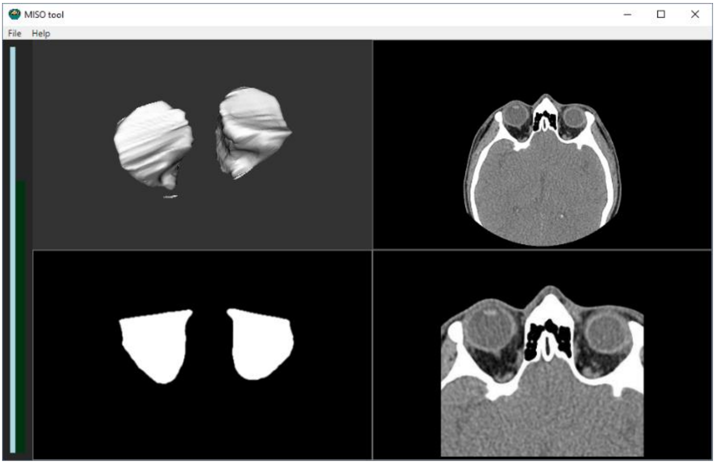


Fig. 5. MISO tool interface

References

[1]. Wagner M.E., Gellrich N.C., Friese K.I. et al. Model-based segmentation in orbital volume measurement with cone beam computed tomography and evaluation against

current concepts. International Journal of Computer Assisted Radiology and Surgery, vol. 11, issue 1, 2016, pp 1–9

[2]. Jean-Franois D, Andreas B. Atlas-based automatic segmentation of head and neck organs at risk and nodal target volumes: a clinical validation. Radiation Oncology, 2013, 8:154

[3]. Yushkevich P.A. Piven J., Hazlett H.C. et al. User-guided 3D active contour segmentation of anatomical structures: significantly improved efficiency and reliability. NeuroImage, vol. 31, issue 3, 2006, pp. 1116-1128

[4]. Doyle, S., Vasseur, F., Dojat, M., Forbes, F. Fully Automatic Brain Tumor Segmentation from Multiple MR Sequences using Hidden Markov Fields and Variational EM. In Procs. of the NCI-MICCAI BRATS, 2013, pp. 18-22

[5]. Cardoso, M.J., Sudre, C.H., Modat, M., Ourselin, S. Template-based multimodal joint generative model of brain data. Lecture Notes in Computer Science, vol. 9123, 2015, pp. 17-29

[6]. H. N. Bharath, S. Colleman, D. M. Sima, S. Van Huffel. Tumor Segmentation from Multimodal MRI Using Random Forest with Superpixel and Tensor Based Feature Extraction. Lecture Notes in Computer Science, vol. 10670, 2018, pp. 463-473.

[7]. Chi-Hoon Lee, Mark Schmidt, Albert Murtha, Aalo Bistritz, Jöerg Sander, Russell Greiner. Segmenting brain tumors with conditional random fields and support vector machines. Lecture Notes in Computer Science, vol. 3765, 2005, pp. 469-478

[8]. Kamnitsas K., Ledig C., Newcombe V.F.J., Simpson J.P., Kane A.D., Menon D.K., Rueckert D., Glocker B. Efficient multi-scale 3DCNN with fully connected CRF for accurate brain lesion segmentation. Medical Image Analysis, vol. 36, 2017, pp. 61-78.

[9]. G. Wang, W. Li, S. Ourselin, T. Vercauteren. Automatic brain tumor segmentation using cascaded anisotropic convolutional neural networks. Lecture Notes in Computer Science, vol. 10670, 2018, pp. 178-190

[10]. Menze B.H., Jakab A., Bauer S. et al. The multimodal brain tumor image segmentation benchmark (BRATS). IEEE Transactions on Medical Imaging, vol. 34, issue 10, 2015, pp. 1993-2024

[11]. Spyridon Bakas, Hamed Akbari, Aristeidis Sotiras et al. Advancing The Cancer Genome Atlas glioma MRI collections with expert segmentation labels and radiomic features. Scientific Data, vol. 4, 2017, Article number: 170117

[12]. Günter Klambauer, Thomas Unterthiner, Andreas Mayr, Sepp Hochreiter. Self-Normalizing Neural Networks. Advances in Neural Information Processing Systems, vol. 30, 2017

[13]. L.K. Hansen and P Salamon. Neural network ensembles. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, issue 10, 1990, pp. 993- 1001

[14]. J. van Doorn. Analysis of deep convolutional neural network architectures. Available at: https://pdfs.semanticscholar.org/6831/bb247c853b433d7b2b9d47780dc8d84e4762.pdf, accessed: 13.06.2018

[15]. Hahnloser R.H., Sarpeshkar R., Mahowald M.A., Douglas R.J., Seung H.S. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature, vol. 405, 2000, pp. 947–951

[16]. O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. Lecture Notes in Computer Science, vol. 9351, 2015, pp. 234-241

[17]. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, vol. 15, issue 1, 2014, pp. 1929-1958

Способы сегментации медицинских изображений

С.А. Мусатян <sabrina.musatian@yandex.ru>
А.В. Ломакин <alexander.lomakin@protonmail.com>
С.Ю. Сартасов <Stanislav.Sartasov@spbu.ru>
Л.К. Попыванов <lev.popyvanov@gmail.com>
И.Б. Монахов <i.monakhov1994@gmail.com>
А.С. Чижова <Angelina.Chizhova@lanit-tercom.com>
Санкт-Петербургский государственный университет,
7/9, Университетская набережная, Санкт-Петербург, 199034

Аннотация. Извлечение различной значимой медицинской информации из КТ и МРТ снимков – это одна из наиболее важных и трудных задач в сфере анализа медицинских изображений. Недостаток автоматизации в этих задачах становится причиной необходимости скрупулезной обработки данных экспертом, что ведет к возможности ошибок, связанных с человеческим фактором. Несмотря на то, что некоторые из методов решения задач могут быть полуавтоматическими, они все еще опираются на человеческие компетенции. Основной целью наших исследований является создание инструмента, который максимизирует уровень автоматизации в задачах обработки медицинских снимков. Наш проект состоит из двух частей: набор алгоритмов для обработки снимков, а также инструменты для интерпретирования и визуализации результатов. В данной статье мы представляем обзор лучших существующих решений в этой области, а также описание собственных алгоритмов для актуальных проблем, таких как сегментация костных глазных орбит и опухолей мозга, используя сверточные нейронные сети. Представлено исследование эффективности различных моделей нейронных моделей при классификации и сегментации для обеих задач, а также сравнительный анализ различных нейронных ансамблей, применяемых к задаче выделения опухолей головного мозга на медицинских снимках. Также представлено наше программное обеспечение под названием «MISO Tool», которое создано специально для подобного рода задач и позволяет выполнять сегментирование тканей с использованием предварительно обученных поставляемых вместе с ПО нейронных сетей, производить различные манипуляции с пиксельными данными DICOM-изображения, а также получать 3D-реконструкция сегментированных областей.

Ключевые слова: глубокие нейронные сети; свёрточные нейронные сети; опухоли мозга; костные глазные орбиты; медицинские изображения.

DOI: 10.15514/ISPRAS-2018-30(4)-12

Для цитирования: Мусатян С.А., Ломакин А.В., Сартасов С.Ю., Попыванов Л.К., Монахов И.Б., Чижова А.С. Способы сегментации медицинских изображений. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 183-194 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(4)-12

Список литературы

[1]. Wagner M.E., Gellrich N.C., Friese K.I. et al. Model-based segmentation in orbital volume measurement with cone beam computed tomography and evaluation against

current concepts. International Journal of Computer Assisted Radiology and Surgery, vol. 11, issue 1, 2016, pp 1–9

[2]. Jean-Franois D, Andreas B. Atlas-based automatic segmentation of head and neck organs at risk and nodal target volumes: a clinical validation. Radiation Oncology, 2013, 8:154

[3]. Yushkevich P.A. Piven J., Hazlett H.C. et al. User-guided 3D active contour segmentation of anatomical structures: significantly improved efficiency and reliability. NeuroImage, vol. 31, issue 3, 2006, pp. 1116-1128

[4]. Doyle, S., Vasseur, F., Dojat, M., Forbes, F. Fully Automatic Brain Tumor Segmentation from Multiple MR Sequences using Hidden Markov Fields and Variational EM. In Procs. of the NCI-MICCAI BRATS, 2013, pp. 18-22

[5]. Cardoso, M.J., Sudre, C.H., Modat, M., Ourselin, S. Template-based multimodal joint generative model of brain data. Lecture Notes in Computer Science, vol. 9123, 2015, pp. 17-29

[6]. H. N. Bharath, S. Colleman, D. M. Sima, S. Van Huffel. Tumor Segmentation from Multimodal MRI Using Random Forest with Superpixel and Tensor Based Feature Extraction. Lecture Notes in Computer Science, vol. 10670, 2018, pp. 463-473.

[7]. Chi-Hoon Lee, Mark Schmidt, Albert Murtha, Aalo Bistritz, Jöerg Sander, Russell Greiner. Segmenting brain tumors with conditional random fields and support vector machines. Lecture Notes in Computer Science, vol. 3765, 2005, pp. 469-478

[8]. Kamnitsas K., Ledig C., Newcombe V.F.J., Simpson J.P., Kane A.D., Menon D.K., Rueckert D., Glocker B. Efficient multi-scale 3DCNN with fully connected CRF for accurate brain lesion segmentation. Medical Image Analysis, vol. 36, 2017, pp. 61-78.

[9]. G. Wang, W. Li, S. Ourselin, T. Vercauteren. Automatic brain tumor segmentation using cascaded anisotropic convolutional neural networks. Lecture Notes in Computer Science, vol. 10670, 2018, pp. 178-190

[10]. Menze B.H., Jakab A., Bauer S. et al. The multimodal brain tumor image segmentation benchmark (BRATS). IEEE Transactions on Medical Imaging, vol. 34, issue 10, 2015, pp. 1993-2024

[11]. Spyridon Bakas, Hamed Akbari, Aristeidis Sotiras et al. Advancing The Cancer Genome Atlas glioma MRI collections with expert segmentation labels and radiomic features. Scientific Data, vol. 4, 2017, Article number: 170117

[12]. Günter Klambauer, Thomas Unterthiner, Andreas Mayr, Sepp Hochreiter. Self-Normalizing Neural Networks. Advances in Neural Information Processing Systems, vol. 30, 2017

[13]. L.K. Hansen and P Salamon. Neural network ensembles. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, issue 10, 1990, pp. 993- 1001

[14]. J. van Doorn. Analysis of deep convolutional neural network architectures. Доступно по ссылке: https://pdfs.semanticscholar.org/6831/bb247c853b433d7b2b9d47780dc8d84e4762.pdf, дата обращения 13.06.2018

[15]. Hahnloser R.H., Sarpeshkar R., Mahowald M.A., Douglas R.J., Seung H.S. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature, vol. 405, 2000, pp. 947–951

[16]. O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. Lecture Notes in Computer Science, vol. 9351, 2015, pp. 234-241

[17]. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, vol. 15, issue 1, 2014, pp. 1929-1958

Применение ассоциативно-семантического препроцессора в интерактивных диалоговых системах на естественном языке

В.Е. Сачков <megawatto@mail.ru>
ФГБОУ ВО МИРЭА – Российский технологический университет,
119454, г. Москва, Проспект Вернадского, д. 78

Аннотация. В статье исследуется возможность применения ассоциативно-семантического препроцессора специальной обработки текста на естественном языке в диалоговых системах. Применение в препроцессоре ассоциаций позволяет абстрагироваться от прямого значения слова и заменить его на набор других слов. Этот эффект имеет и обратное действие: по набору слов (ассоциаций) можно восстановить искомое слово, что позволяет человеку формировать запрос на естественном языке, не зная ключевых слов или терминов той или иной предметной области, но при этом получать нужный ему результат. При семантической обработке текста с использованием ассоциаций совершенно не важен порядок слов и их количество, что позволяет человеку общаться с машиной, не формируя фразы специальным образом, так как интерактивная диалоговая система сама обработает запрос, очистив его от всего лишнего. Применение специального текстового препроцессора на основе ассоциативно-семантической обработки текста позволяет наделить интерактивные диалоговые системы способностью к пониманию темы диалога машины с пользователем, улучшить взаимодействие путем общения на естественном языке, а также упростить процесс создания и разработки диалоговых систем.

Ключевые слова: семантика; диалоговая система; метрика EMD; поисковая система; ассоциации

DOI: 10.15514/ISPRAS-2018-30(4)-13

Для цитирования: Сачков В.Е. Применение ассоциативно-семантического препроцессора в интерактивных диалоговых системах на естественном языке. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 195-208. DOI: 10.15514/ISPRAS-2018-30(4)-13

1. Введение

Текст на естественном языке понятен пользователю, но для машины он представляется не более чем набором закодированных символов. Для извлечения ценных данных машине необходимо решить множество задач по обработке текста на естественном языке. Такими задачами занимается специальный раздел прикладной лингвистики «Автоматическая обработка текстов на естественном языке» (Natural Language Processing, NLP). На сегодняшний день существует большое разнообразие задач обработки естественного языка, наиболее распространенными среди которых являются следующие [1]:

- поиск фрагментов текста – разделение текста на различные элементы разных типов: слова, предложения, абзацы и т. д.;
- поиск предложений (Sentence Boundary Disambiguation, SBD) – определение границ предложения;
- поиск именованных объектов (Named entity recognition, NER) – механизм поиска адресов, названий, имен, дат, или любых других именованных сущностей;
- определение частей речи (Parts of speech, POS) – классификация элементов текста на уровне предложения; предложение может быть разделено на отдельные слова и словосочетания по таким категориям, как существительные, глаголы, наречия, предлоги и т. д.;
- классификация текстов и документов – цель данной классификации в присвоении меток фрагментам, найденным в текстах и документах;
- выделение взаимоотношений – выявление связей между словами или словосочетаниями для построения семантического дерева.

Одной из приоритетных задач автоматической обработки текста, является семантический анализ – понимание содержательной и смысловой части текста. При создании диалоговых систем на естественном языке широко применяется семантика, поскольку от этих систем в первую очередь требуется понимание запроса пользователя. Эталонным примером может служить первая диалоговая система «Eliza», созданная в 1966 году и имитирующая диалог с психоаналитиком [2]. Для своего времени программа «Eliza» была прорывом, но ее алгоритм основывался на простом перефразировании вопросов и ей было далеко до понимания «смысла».

Современный пример развития диалоговых систем можно представить программой «A.L.I.C.E» [3] – это виртуальный собеседник, способный вести диалог на естественном языке. Основой «A.L.I.C.E» является язык разметки искусственного интеллекта (Artificial Intelligence Markup Language, AIML), более детально рассматриваемый в статье далее.

При создании «Eliza», «A.L.I.C.E» и тому подобных систем ключевым моментом всегда оставалась потребность в понимании «смысла» запроса пользователя на естественном языке, т.е. потребность в инструменте, способном выявить из текста информацию, о чем пользователь спрашивает систему, и привести эту информацию в формальный и понятный вычислительной машине вид. Разработка такого инструмента до сих остается актуальной проблемой.

2. Анализ задачи

Самым сложным этапом автоматической обработки текста на естественном языке считается содержательный анализ. Для его успешного выполнения необходимы знания о том, что такое значения слов и предложений, как эти значения описать формально, как представлять и хранить смысловое содержание текста в памяти компьютера, как производить операции со значениями, переводить значения с естественного языка на формальный язык и обратно. Ответы на эти и многие другие вопросы дает компьютерная семантика, в ведении которой находится разработка моделей семантического уровня естественного языка [4].

Применение интерактивных диалоговых систем (ИДС) значительно упрощает взаимодействие вычислительной машины с пользователем за счет того, что общение между ними происходит на естественном языке. Пользователь такой системы не обязан обладать специальными навыками и знаниями, может вести с ИДС осмысленный диалог, что обеспечивает удобство использования системы и повышает уровень доверия пользователя к этой системе. Для реализация таких возможностей в ИДС часто требуется разработка сложных интеллектуальных систем, основанных на базах знаний, правилах, словарях и т.д.

Современная тенденция развития ИДС по-прежнему предполагает использование значительных инженерных и экспертных знаний. В статье [5] представлен обзор такого подхода с детальным разбором наборов данных, корпусов документов на естественном языке, с помощью которых можно создавать диалоговые системы на базе моделей машинного обучения для разных тематик.

Одним из важных свойств диалоговых систем является корректность сформированного ответа на вопрос пользователя, и очень важно уметь оценивать ответ, выдаваемый пользователю на естественном языке. Конечном результатом должна быть оценка удовлетворённости пользователя [6]. В работе [7] проведены сравнения нескольких машинных метрик оценки результатов диалоговой системы с оценками человека.

Генерация естественного языка является критическим компонентом разговорного диалога и оказывает значительное влияние как на удобство использования, так и на воспринимаемое качество. В большинстве случаев используются правила и эвристики, генерируются жесткие и стилизованные ответы без естественного изменения человеческого языка. В работе [8] описывается генератор ответов на основе семантически контролируемой структуры нейронной сети «Long Short-term Memory» (LSTM) [9], позволяющий приблизить машинные ответы к естественному языку. Другим примером построения диалоговой системы может служить работа [10], где используются большие наборы данных для тренировки рекуррентных нейронных сетей.

Чтобы научить компьютер понимать естественный язык, его требуется оснастить механизмом, позволяющим получать, извлекать и обрабатывать содержание документа, понимать смысл слов, словосочетаний и отдельных предложений. Для этого требуется применять методы компьютерной семантики, и компьютеру нужна некая структура, чтобы хранить соответствующие знания.

Один из подходов основан на использовании концепции Semantic Web [11], на основе которой была разработана модель документов RDF [12], позволяющая хранить семантические структуры. Существенной проблемой модели RDF является сложная структура онтологий, а у языка SPARQL, предложенного в качестве стандарта для работы с RDF, имеется ряд существенных синтаксических и семантических недостатков. Кроме того, концепция Semantic Web получила ограниченное распространение среди разработчиков, поскольку приходилось создавать два одинаковых по содержанию документа, но один для «людей», а второй для «машины».

Другой современный подход к пониманию семантики текста, поиску близких по смыслу слов и определению тематики текста основан на обучении моделей нейронных сетей. Существует два конкурирующих подхода, один из которых основан на частотном вхождение в корпус документов, а другой – на модели прогнозирования. Популярным представителем первого подхода является латентно-семантический анализ (Latent semantic analysis, LSA) [13], представитель второго – набор алгоритмов word2vec [14].

Модель LSA основана на частотных подсчетах, где аналогичные слова имеют одинаковые значения в разных документах. LSA широко используется в поисковых системах для индексирования и поиска близких по смыслу слов и документов. Но у этой модели имеется ряд недостатков, в частности, разрежённость данных, игнорирование семантических ассоциаций между словами. Кроме того, модель плохо работает с большими массивами данных из-за большого потребления памяти при проведении расчетов и проигрывает word2vec по качеству результатов [15].

В модели word2Vec слово представляется в виде вектора, а сама модель предсказывает набор векторов, ближайших к исходному, по дистрибутивным признакам. Преимуществом и одновременно недостатком модели является необходимость в обучении на большом корпусе данных, при этом она более экономна по вычислительным ресурсам. Хотя модель word2Vec показывает результаты, лучшие, чем LSA, ей свойственна проблема частотного перекрытия слов, что порождает семантическую неоднозначность.

Рассмотренные модели обладают рядом других преимуществ и недостатков. Более детальное сравнение приводится, например, в работе [16].

Подводя итоги, можно заключить, что для разработки современной ИДС необходимо решение, которое учитывает семантические ассоциации, основано на прогнозируемой модели сходства слов, решает задачу частотного перекрытия, а также обладает алгоритмом сравнения двух документов на естественном языке. С учетом этих требований было разработано специальное программное обеспечение «Ассоциативно-семантический текстовый препроцессор». В статье описываются результаты его применения при создании современных ИДС.

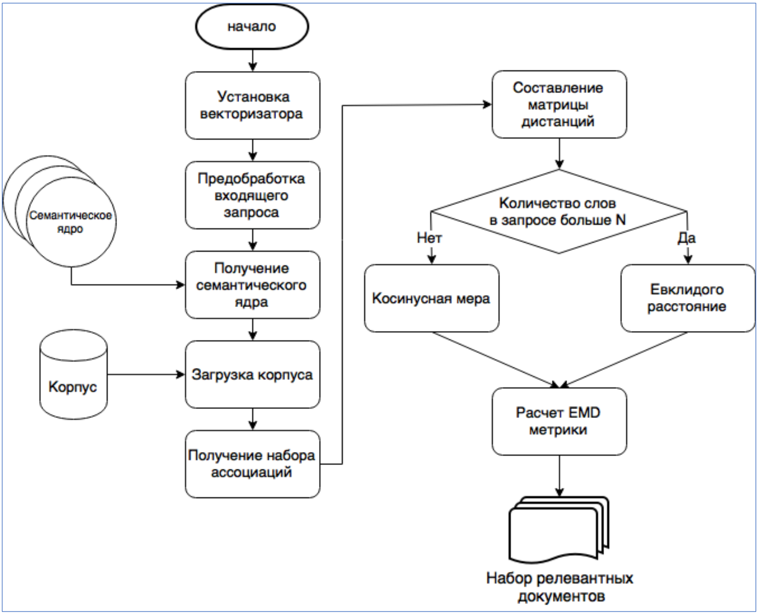


Рис. 1. Алгоритм ассоциативно-семантического поиска
Fig. 1. Algorithm of associative-semantic search

3. Ассоциативно-семантический текстовый препроцессор

Ассоциативно-семантический текстовый препроцессор (АСТП) – предназначен для предварительной нормализации текстов с целью преобразования в наборы ассоциативных семантических векторов с заданной смысловой аннотацией. Дополнительно поддерживаются функции обработки естественного языка в процессах взаимодействия естественного языка и языков компьютерных систем и роботизированных устройств (токенизация и стеммизация) для последующей обработки средствами NLP корпуса документов для проверки гипотез, обучения и статистического лингвистического анализа.

Препроцессор АСТП может использоваться как для встраивания в пакеты прикладных программ, так и в качестве самостоятельного приложения. На рис. 1 показан один из алгоритмов АСТП, отвечающий за семантический поиск близких по смыслу текстов.

Рассмотрим наиболее важные элементы алгоритма.

- Семантическое ядро – это векторное пространство семантического поля, в котором производится поиск близких по смыслу слов и ассоциаций. Оно представляет упорядоченный набор слов или словосочетаний, наиболее точно характеризующий предметную область, вид деятельности или предмет, и позволяет создавать наиболее релевантные поисковые запросы. Семантическое ядро имеет центральное ключевое слово, как правило, высокочастотное, и все остальные ключевые слова в нем ранжируются по мере убывания частоты совместного использования с центральным запросом в общей коллекции документов [17].
- Корпус – это набор специально подготовленных документов, среди которых производится семантический поиск. В наших экспериментах корпус собирался с новостного портала «РИА НОВОСТИ» (https://ria.ru), с помощью веб-скраппера (web-scraping). Корпус состоит из 250 тысяч документов на естественном языке по 13 основным разделам сайта за 3 года новостных публикаций портала.
- Модуль оценки ассоциаций – это модуль, который производит оценку и фильтрацию найденных ассоциаций в семантическом ядре.
- Метрика EMD (Earth Mover’s Distance) – это метод оценки несходства между двумя многомерными распределениями в некотором пространстве признаков, для которого задана дистанционная мера между одиночными признаками [18]. Метрика EMD вычисляет минимальную стоимость изменений или работы, требуемой для преобразования одного документа в другой. Вычисление значения EMD базируется на решении транспортной задачи.
- Матрица расстояний –матрица весов каждого слова в документе для расчёта метрики EMD

Упрощенный алгоритм ассоциативно-семантического поиска выполняет следующие шаги:

1. устанавливается тип вектора, в котором будут храниться данные;
2. производиться предобработка текста: токенизация, стеммизация, лемматизация, удаление стоп слов и т.д.;
3. загружается необходимое семантическое ядро в зависимости от темы входящего сообщения; выбор семантического ядра осуществляет обученная модель по алгоритму мультиноминального байесовского классификатора [19];
4. загружается необходимый корпус документов, в котором производится поиск;

5. для поискового запроса рассчитывается матрица расстояний в векторном пространстве семантического ядра;
6. для каждого документа корпуса создается вектор ассоциаций;
7. рассчитывается дистанция EMD;
8. возвращается отсортированный ревалентный список документов, удовлетворяющих поисковому запросу.

```
<aiml version="1.0.1" encoding="UTF-8">

    <!--HELLO-->

    <category>
        <pattern>ПРИВЕТ</pattern>
        <template>
            <random>
                <li>Привет!</li>
                <li>Добрый день!</li>
                <li>Рада Вас видеть!</li>
            </random>
            <random>
                <li>Как вас зовут?</li>
                <li>Ваше имя?</li>
                <li>Как мне вас называть?</li>
            </random>
        </template>
    </category>

    <category>
        <pattern>*</pattern>
        <that>_ КАК ВАС ЗОВУТ</that>
        <template>
            Я запомню вас как
            <set name="user">
                <star/>
            </set>
        </template>
    </category>
```

Рис. 2. Шаблон AIML
Fig. 2. AIML Template

В качестве основы для создания ИДС был выбран специальный стандарт языка шаблонной разметки AIML AIML – это язык разметки искусственного интеллекта, основанный с XML и позволяющий создавать виртуальных собеседников [20]. Использование этого языка разметки ускоряет создание диалоговой системы с применением препроцессора АСТП для ассоциативно-семантического поиска. Пример документа AIML представлен на рис. 2.

Для создания AIML документа, который позволил бы ввести диалог ассоциативно-поисковый системы с пользователем на естественном языке, были разработаны следующие смысловые блоки:

- приветствие – этот блок отвечает за начало работы с пользователем, здесь диалоговая система запоминает имя пользователя для дальнейшего использования в диалоге;
- о себе – в этом блоке диалоговая система рассказывает о себе и своих возможностях;
- ассоциации – этот блок отвечает за обработку запросов, связанных с поиском ассоциаций, с использованием препроцессора АСТП;
- поиск – этот блок отвечает за ассоциативный поиск;
- ответ по умолчанию – этот блок срабатывает только в том случае, когда ни один из предыдущих блоков не смог обработать запрос пользователя.

Упрощенная схема ведения диалога на естественном языке представлена на рис. 3.

ИДС значительно упрощает поиск близких по смыслу текстов и ассоциаций, так как позволяет пользователям общаться с поисковой системой на естественном языке, увеличивая количество формулировок запроса, которые можно использовать. Применение семантических ядер позволяет ИДС довольно точно определять тематику диалога с пользователем, и если тематика не определяется однозначно, то ИДС задает уточняющий вопрос, корректирующий подбор семантического ядра. После определения тематики диалога ИДС способна выполнить два действия.

1. Поиск ассоциаций – поиск производится путем вычисления семантического расстояния в семантическом ядре. Вычисление базируется на дистрибутивной семантике – области научных исследований, занимающейся вычислением степени семантической близости между лингвистическими единицами на основе их дистрибуционных признаков в больших массивах лингвистических данных [21]
2. Поиск близких по смыслу текстов: происходит по алгоритму, рассмотренному ранее, с вычислением значений метрики EMD.

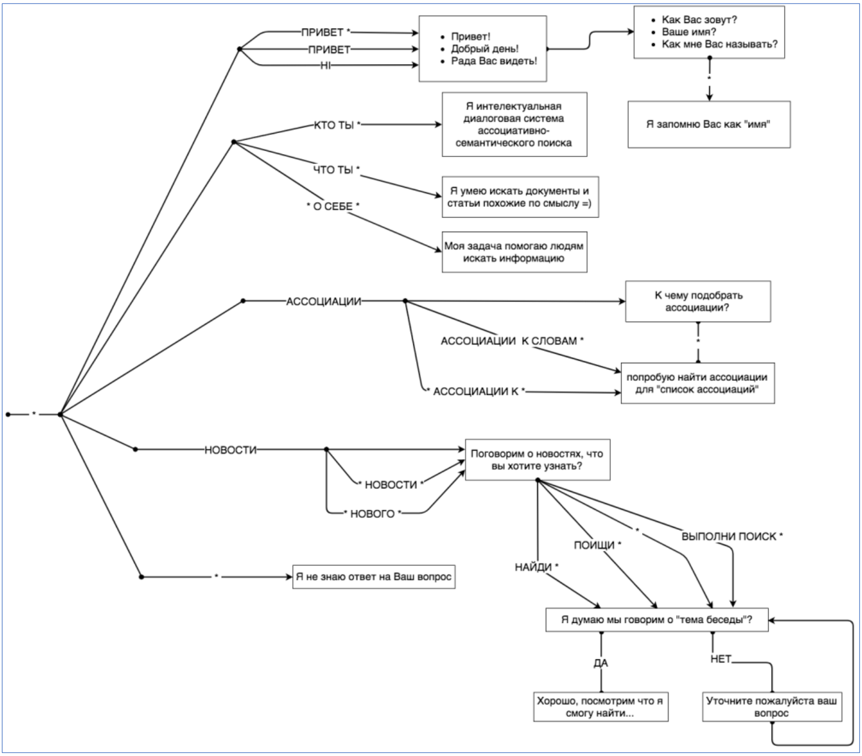


Рис. 3. Схема ведения диалога на естественном языке с пользователем
Fig. 3. Diagram of dialogue in natural language with the user

Примером применения алгоритмов АСТП с языком разметки AIML может служить работа небольшого приложения для поиска ассоциаций в семантических ядрах новостного корпуса документов. Соответствующий диалог представлен на рис. 4.

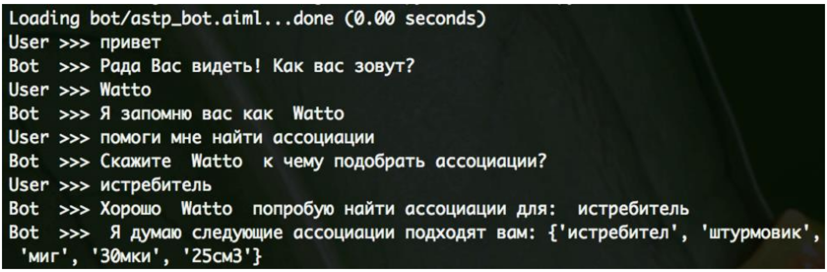


Рис. 4. Разговор с диалоговой системой
Fig. 4. General conversation with the dialogue system

В завершение продемонстрируем работу ИДС по поиску близких по смыслу новостей из корпуса документов на тему «Крушения самолета». Диалог представлен на рис. 5.

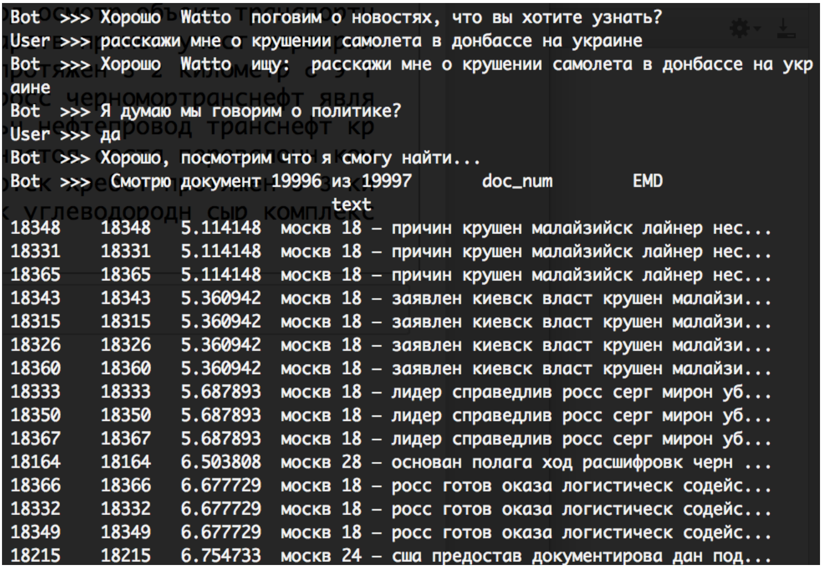


Рис. 5. Поиск документов в ИДС

Fig. 5. Searching documents in an interactive dialog system

4. Заключение

Как показывают исследования возможности применения специализированного программного обеспечения АСТП в ИДС, АСТП повышает уровень взаимодействия пользователя и машины, так как позволяют вести диалог на естественном языке. В отличие, например, от модели RDF, применение данного программного обеспечения не требует большой сложности в составлении онтологий и сложных поисковых запросов по корпусам документов, не обладают такой же большой сложностью в реализации и обучении, что позволяет наделить ИДС минимально необходимым пониманием темы диалога для решения типовых задач, не привлекая большие вычислительные и интеллектуальные ресурсы.

Использование семантики, ассоциаций и современных методов компьютерной обработки текстов на естественном языке позволяет улучшить качество поиска, а также увеличить степень понимания диалога машины с человеком и наделить компьютер дополнительными инструментами для общения на естественном языке.

Благодарности

Работа выполнена за счет финансирования Министерством образования и науки РФ конкурсной части государственных заданий высшим учебным заведениям и научным организациям по выполнению инициативных научных проектов. Номер проекта 28.2635.2017/ПЧ, наименование «Разработка моделей стохастической самоорганизации слабоструктурированной информации и реализации памяти при прогнозировании новостных событий на основе массивов естественно-языковых текстов».

Список литературы

[1]. Сачков В.Е., Гильмутдинова Е.Ф., Матяш Е.Д., Акимов Д.А. Обработка и компьютерный анализ текста на естественных языках. Современная наука: актуальные проблемы теории и практики, серия Естественные и технические науки, № 12, 2016 г., стр. 57-64

[2]. Weizenbaum J. Computer Power and Human Reason: From Judgment to Calculation. Freeman and Company, New York, 1976, 281 p.

[3]. Wallace R. S. The Anatomy of A.L.I.C.E. In Parsing the Turing Test. Springer, 2009, pp. 181-210

[4]. Прикладная и компьютерная лингвистика. Под ред. И. С. Николаева, О. В. Митрениной, Т. М. Ландо. М., ЛЕНАНД, 2016, 320 стр.

[5]. Serban I. V., Lowe R., Henderson P., Charlin L., Pineau J. A Survey of Available Corpora for Building Data-Driven Dialogue Systems [электронный ресурс] URL: https://arxiv.org/abs/1512.05742 (дата обращения: 01.06.18)

[6]. Walker M., Litman D., Kamm C., Abella A. Paradise: A framework for evaluating spoken dialogue agents, In Proc. of the Eighth Conference on European chapter of the Association for Computational Linguistics, Spain, 1997, pp. 271–280

[7]. Chia-Wei Liu, Lowe R., Serban I. V., Noseworthy M., Charlin L., Pineau J. How NOT to Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation [электронный ресурс] URL: https://arxiv.org/abs/1603.08023 (дата обращения: 01.06.18)

[8]. Henderson M., Thomson B., Williams J. The Second Dialog State Tracking Challenge [электронный ресурс] URL: http://camdial.org/~mh521/dstc/downloads/handbook.pdf (дата обращения: 01.06.18)

[9]. Hochreiter S., Schmidhuber J. Long Short-Term Memory. Neural Computation, vol. 9, issue 8, 1997, pp.1735-1780

[10]. Serban I. V., Sordoni A., Bengio Y., Courville A., Pineau J. Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models [электронный ресурс] URL: https://arxiv.org/abs/1507.04808 (дата обращения: 01.06.18)

[11]. W3C Semantic web activity [электронный ресурс]. URL: https://www.w3.org/2001/sw/ (дата обращения: 25.06.2018)

[12]. Среда Описания Ресурса (RDF): Понятия и Абстрактный Синтаксис [электронный ресурс]. URL: https://www.w3.org/2007/03/rdf_concepts_ru (дата обращения: 20.02.2017)

[13]. Landauer T., Foltz P. W., Laham D. Introduction to Latent Semantic Analysis. Discourse Processes, vol. 25, issue 2-3, 1998, pp. 259–284

[14]. Tomas Mikolov, Quoc V. Le, Ilya Sutskever Exploiting Similarities among Languages for Machine Translation [электронный ресурс] URL: https://arxiv.org/pdf/1309.4168.pdf (дата обращения: 20.04.2017)

[15]. Levy O., Golberg Y., Dagan I. Improving Distributional Similarity with Lessons Learned from Word Embeddings [электронный ресурс] URL: http://www.aclweb.org/anthology/Q15-1016 (дата обращения: 01.06.2018)

[16]. Altszyler E., Sigman M., Ribeiro S., Slezak D.F. Comparative study of LSA vs Word2vec embeddings in small corpora: a case study in dreams database [электронный ресурс] URL: https://arxiv.org/pdf/1610.01520.pdf (дата обращения: 01.06.2018)

[17]. Семантическое ядро [электронный ресурс] URL: https://ru.wikipedia.org/wiki/Семантическое_ядро (дата обращения: 25.05.2017)

[18]. Rubner Y., Tomasi C., Guibas L. J. A metric for distributions with applications to image databases. In Proc. of the Sixth International Conference on Computer Vision, 1998, pp. 59-66

[19]. AIML: Artificial Intelligence Markup Language [электронный ресурс] URL: http://www.alicebot.org/aiml.html (дата обращения: 13.06.2017)

[20]. Jason D. M., Rennie, Shih L., Teevan J., Karger D. R. Tackling the Poor Assumptions of Naive Bayes Text Classifiers. In Proc. of the Twentieth International Conference on Machine Learning (ICML), 2003, pp. 616-623

[21]. Морозова Ю.И. Построение семантических векторных пространств различных предметных областей. Третья школа молодых ученых ИПИ РАН. Сборник докладов, 2012, стр. 4-11

The use of associative semantic preprocessor in the interactive dialogue systems in natural language

V.E. Sachkov <megawatto@mail.ru>
MIREA - Russian Technological University,
78 Vernadsky Avenue, Moscow 119454

Abstract. The article explores the possibility of using an associative-semantic preprocessor for special text processing in natural language. The use of associations allow to abstract from the direct meaning of a word and to replace it with a set of other words. This has also the opposite effect: by typing words (associations) a person is able to restore the search word, which allows to form a query in a natural language without knowing the keywords or terms of a particular domain but at the same time to receive the required result, in contrast to systems oriented to frequency occurrences of words. In the semantic processing of text using associations, the order of words and their number are not important, which allows a person to communicate with the machine without formulating phrases in a special way, since the interactive dialog system itself will process the request clearing everything else. The use of a special text preprocessor based on the associative-semantic processing of text allows interactive systems to be able to understand the topic of the machine's dialogue with the user, improve interaction by communicating in a natural language, and also to simplify the process of system creation and development.

Keywords: semantics; dialogue System; EMD; search engine; associations

DOI: 10.15514/ISPRAS-2018-30(4)-13

For citation: Sachkov V.E. The use of associative semantic preprocessor in the interactive dialogue systems in natural language. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 4, 2018. pp. 195-208 (in Russian). DOI: 10.15514/ISPRAS-2018-30(4)-13

References

[1]. Sachkov V.E, Gilmutdinova E.F, Matyash E.D, Akimov D.A. Processing and computer analysis of the text in natural languages. Sovremennaja nauka: aktual'nye problemy teorii i praktiki, serija Estestvennye i tehnicheskie nauki [Journal of Contemporary Science: Actual Problems of Theory and Practice, Series of Natural and Technical Sciences], № 12. 2016. pp. 57-64 (in Russian)

[2]. Weizenbaum J. Computer Power and Human Reason: From Judgment to Calculation. Freeman and Company, New York, 1976, 281 p.

[3]. Wallace R. S. The Anatomy of A.L.I.C.E. In Parsing the Turing Test. Springer, 2009, pp. 181-210

[4]. Applied and Computational Linguistics, I.S Nikolaeva, O.V Mitrenina, T.M Lando, eds. Moscow, LENAND, 2016, 320 p. (in Russian)

[5]. Serban I. V., Lowe R., Henderson P., Charlin L., Pineau J. A Survey of Available Corpora for Building Data-Driven Dialogue Systems [online] URL: https://arxiv.org/abs/1512.05742 (accessed at: 01.06.18)

[6]. Walker M., Litman D., Kamm C., Abella A. Paradise: A framework for evaluating spoken dialogue agents, In Proc. of the Eighth Conference on European chapter of the Association for Computational Linguistics, Spain, 1997, pp. 271–280

[7]. Chia-Wei Liu, Lowe R., Serban I. V., Noseworthy M., Charlin L., Pineau J. How NOT to Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation [online] URL: https://arxiv.org/abs/1603.08023 (accessed at: 01.06.18)

[8]. Henderson M., Thomson B., Williams J. The Second Dialog State Tracking Challenge [online] URL: http://camdial.org/~mh521/dstc/downloads/handbook.pdf (accessed at: 01.06.18)

[9]. Hochreiter S., Schmidhuber J. Long Short-Term Memory. Neural Computation, vol. 9, issue 8, 1997, pp.1735-1780

[10]. Serban I. V., Sordoni A., Bengio Y., Courville A., Pineau J. Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models [online] URL: https://arxiv.org/abs/1507.04808 (accessed at: 01.06.18)

[11]. W3C Semantic web activity [online]. URL: https://www.w3.org/2001/sw/ (accessed at: 25.06.2018)

[12]. Resource Description Framework (RDF): Concepts and Abstract Syntax [online]. URL: https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (accessed at 20.02.2017)

[13]. Landauer T., Foltz P. W., Laham D. Introduction to Latent Semantic Analysis. Discourse Processes, vol. 25, issue 2-3, 1998, pp. 259–284

[14]. Tomas Mikolov, Quoc V. Le, Ilya Sutskever. Exploiting Similarities among Languages for Machine Translation [online] URL: https://arxiv.org/pdf/1309.4168.pdf (accessed at: 20.04.2017)

[15]. Levy O., Golberg Y., Dagan I. Improving Distributional Similarity with Lessons Learned from Word Embeddings [online] URL: http://www.aclweb.org/anthology/Q15-1016 (accessed at: 01.06.2018)

[16]. Altszyler E., Sigman M., Ribeiro S., Slezak D.F. Comparative study of LSA vs Word2vec embeddings in small corpora: a case study in dreams database [onine] URL: https://arxiv.org/pdf/1610.01520.pdf (accessed at: 01.06.2018)

[17]. Semanticheskoe yadro [Semantic kernel] [online] URL: https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D0%BD%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D1%8F%D0%B4%D1%80%D0%BE, accessed at: 25.05.2017 (in Russian)

[18]. Rubner Y., Tomasi C., Guibas L. J. A metric for distributions with applications to image databases. In Proc. of the Sixth International Conference on Computer Vision, 1998, pp. 59-66

[19]. AIML: Artificial Intelligence Markup Language [online] URL: http://www.alicebot.org/aiml.html (accessed at: 13.06.2017)

[20]. Jason D. M., Rennie, Shih L., Teevan J., Karger D. R. Tackling the Poor Assumptions of Naive Bayes Text Classifiers. In Proc. of the Twentieth International Conference on Machine Learning (ICML), 2003, pp. 616-623

[21]. Morozova Yu.I. The construction of semantic vector spaces of different subject domains. In Proc. of the Third School of Young Scientists of the IPI RAS, 2012, pp. 4 – 11 (in Russian)

Об онлайновых алгоритмах для задач упаковки в контейнеры и полосы, их анализе в худшем случае и в среднем

[1,2] *Д.О. Дазарев <dennis810@mail.ru>*
[1,2] *Н.Н. Кузюрин <nnkuz@ispras.ru>*
[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский физико-технический институт, 141700, Московская область, г. Долгопрудный, Институтский пер., 9*

Аннотация. В работе рассмоторены онлайновые алгоритмы для классических задач упаковки Bin Packing и Strip Packing и их обобщений: задач Multidimensional Bin Packing, Multiple Strip Packing и задаче об упаковке в полосы различной ширины. Для последней задачи описан анализ в худшем случае; для остальных задач приведен как анализ в худшем случае, так и анализ в среднем (вероятностный анализ). Рассмотрены лучшие известные нижние и верхние оценки. Приведены основные алгоритмы и описаны методы их анализа.

Ключевые слова: Bin Packing; Multidimensional Bin Packing; Strip Packing; Multiple Strip Packing; задача об упаковке в полосы различной ширины; вероятностный анализ; анализ в худшем случае

DOI: 10.15514/ISPRAS-2018-30(4)-14

Для цитирования: Лазарев Д.О., Кузюрин Н.Н. Об онлайновых алгоритмах для задач упаковки в контейнеры и полосы, их анализе в худшем случае и в среднем. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 209-230. DOI: 10.15514/ISPRAS-2018-30(4)-14

1. Введение

В последние годы заметно повысился интерес к задачам оптимизации в различных производственных и логистических процессах [1,2,3,11]. Для популярных в настоящее время задач анализа больших данных часто используются облачные вычисления, которые также требуют решения задач оптимизации. Важную роль в такого рода задачах оптимизации играет теория расписаний и, в частности, различные классы задач упаковки [4,5].

Интерес к задачам упаковки всегда стимулировался их многочисленными практическими приложениями. Так, одномерная задача упаковки в контейнеры (bin packing) возникла в силу потребности решения задач раскроя и перевозки материалов. Двумерная задача упаковки в контейнеры хорошо моделировала задачу оптимизации размещения объектов, например, автомобилей, в вагонах, паромах и т.п., а также размещения микросхем на платах в задачах построения СБИС. Трехмерная задача упаковки учитывала еще одно измерение и хорошо описывала оптимизацию размещения трехмерных объектов на складах и т.п.

В настоящее время в силу быстрого роста популярности распределенных вычислений, широкого распространения вычислительных кластеров, грид-технологий а также облачных вычислений этот интерес к задачам упаковки возрастает в связи с новыми приложениями: задачами управления ресурсами распределенных вычислительных систем, развития техники облачных вычислений [6,7,8,12,35,41]. При этом возникают новые классы задач упаковки, в частности, задачи упаковки прямоугольников в полосу и несколько полос, задачи упаковки приложений в виртуальные машины и контейнеры и т.п. В данной работе мы ставим целью описать классические результаты, касающиеся различных задач упаковки, а также привести ряд новых результатов, полученных в самое последнее время.

2. Постановка задачи

2.1 Bin Packing

Определим задачу Multidimensional Bin Packing Problem (Multidimensional BPP), или задачу об упаковке в контейнеры размерности n в постановке упаковки в ящик (Box Packing): дан набор d-мерных открытых прямоугольных параллелипипедов, длины сторон которых не превосходят 1. Требуется упаковать параллелипипеды без вращений и пересечений в как можно меньшее число d-мерных кубов с единичной стороной.

Например, в случае одномерной задачи Bin Packing (BP) требуется распределить объекты в минимальное количество контейнеров так, чтобы суммарный вес объектов в каждом контейнере не превышал 1.

Задача является NP-трудной, поэтому будем рассматривать приближённое решение. Будем рассматривать случай онлайновой упаковки, когда алгоритм получает параллелипипеды поочередно, и размещение каждого следующего объекта не влияет на положение предыдущих. Существуют два способа анализа таких алгоритмов.

- Анализ в худшем случае, или *Worst Case* анализ.

  Здесь эффективность алгоритма A на наборе параллелипипедов σ оценивается через асимптотическую мультипликативную точность

  $$R_A^{\infty} = \limsup_{n \to \infty} \sup_{\sigma} \left\{ \frac{cost_A(\sigma)}{cost(\sigma)} \middle| cost(\sigma) \geq n \right\},$$

  где $cost_A(\sigma)$ – число кубов, занятых параллелипипедами из σ, при их упаковке алгоритмом A; $cost(\sigma)$- минимальное возможное число занятых кубов при упаковке набора параллелепипедов.

- Анализ в среднем, или *Average Case* анализ.

  В предположении, что длины сторон параллелипипедов имеют распределение F, чаще всего – равномерное распределение на некотором отрезке $[0, u]$, $u \le 1$, нужно оценить математическое ожидание суммарного объема площади контейнеров, не заполненной параллелипипедами, $W_A^n = E_F V_A^n$, после выпадения (не более чем) n параллелипипедов

При анализе в худшем случае исследуются лишь алгоритмы, которым неизвестно количество объектов до выпадения последнего из них. При анализе в среднем можно анализировать эффективность алгоритмов, когда число параллелепипедов известно до начала работы (*closed-end*) и когда число параллелипипедов становится известным лишь тогда, когда все они выпали (*open-end*). В случае *open-end* W_A^n вычисляется после выпадения ровно n параллелипипедов.

В [24] Шором было ослаблено условие *open-end* и было дано следующее описание работы алгоритмов *open-end по Шору*:

- натуральное число n известно алгоритму до начала работы;
- выбирается случайно натуральное $k, 1 \le k \le n$;
- на вход подается k объектов со случайными длинами сторон и после них – символ останова.

Оценивается математическое ожидание W_A^n объёма контейнеров, не заполненного после упаковки всех k объектов.

2.2 Strip Packing и Multiple Strip Packing

В зависимости от количества полос задачи разделяются на следующие:

- Strip Packing(SP): дана полубесконечная полоса единичной ширины;
- Multiple Strip Packing(MSP): дан набор полубесконечных полос $C = \{C_1, \dots, C_m\}$ единичной ширины.

В эти полосы требуется упаковать без вращений и пересечений n открытых прямоугольников $T = \{T_1, \dots, T_n\}$, минимизируя при этом высоту упаковки. Высота упаковки – координата самой высокой верхней стороны одного из прямоугольников. Исследуются онлайновые алгоритмы упаковки, то есть алгоритмы, получающие прямоугольники из T последовательно, и размещающие T_i до получения $T_{i+1}, \dots, T_n$ для любого $i \le n - 1$.

Как и в случае задачи BP, существует два способа анализа алгоритмов упаковки:

- Анализ в худшем случае, или *Worst Case* анализ.

  Здесь эффективность алгоритма A на наборе прямоугольников T

оценивается через его асимптотическую мультипликативную точность

$$R_A^\infty = \lim_{n \to \infty} sup \; \sup_T \left\{ \frac{H_A(T)}{H_O(T)} \middle| H_O(T) \geq n * h_{max} \right\},$$

где $H_A(T)$ – высота упаковки прямоугольников из T при их упаковке алгоритмом A; $H_O(T)$- минимальная возможная высота упаковки прямоугольников из T, h_{max}-максимальная высота прямоугольника. Анализ проводится в предположении, что h_{max} известно алгоритму до начала работы

- Анализ в среднем, или *Average Case* анализ.

  В предположении, что длины и высоты прямоугольников имеют распределения F_i и F_w, чаще всего – равномерные распределения на некоторых отрезках $[0, u]$ и $[o, v]$ соответсвенно, нужно оценить математическое ожидание суммарной площади части полос от основания до высоты упаковки, не заполненной прямоугольниками $W_A^n = E_{F_l,F_w} S_A^n$, после выпадения n прямоугольников.

При оценке в среднем эффективность алгоритмов анализируется, как и для задачи BP, в *open-end* и *close-end* случаях.

3 Задача Bin Packing

Задача Bin Packing или задача упаковки в контейнеры – одна из первых известных NP-трудных в сильном смысле задач [9]. Для нее было составлено множество приближённых алгоритмов для анализа в худшем и в среднем случаях [10].

Опишем важные методы онлайновой упаковки.

First Fit. Каждый следующий поступивший объект попадает в последний созданный контейнер, в который он помещается. Если же он не помещается ни в один контейнер, то создаётся новый контейнер, в котором объект и оказывается.

Best Fit. Каждый следующий объект попадает в наиболее плотно заполненный контейнер, в который он может поместиться. Если объект никуда не помещается, то для его упаковки создаётся новый контейнер.

3.1 Анализ в худшем случае

Следуя классической работе [11], покажем, что асимптотическая точность алгоритма First Fit $R_{FF}^\infty = \frac{17}{10}$.

Пусть на вход алгоритму First Fit поступило в порядке возрастания индексов k объектов $a_1, \ldots, a_k$ размерами $s(a_1), \ldots, s(a_k)$, образующих набор σ. Алгоритм

First Fit их размещает в контейнеры $B_1, \dots, B_n$; номера контейнеров возрастают в порядке их создания.

Определение: грубостью α_i i-го контейнера B_i назовём максимальную незаполненную длину контейнера с номером, меньшим i.

Ниже $cost_A(\omega)$ обозначает число контейнеров, в которые алгоритм A размещает объекты из набора ω, а OPT – оптимальный алгоритм.

Теорема 1 (Johnson, Demers, Ullman, Garey, Graham, 1974) [11]. Для любого набора σ, верно, что $cost_{FF}(\sigma) \le \frac{17}{10} cost_{OPT}(\sigma) + 2$.

В доказательстве используется весовая функция $W(a)$. Для доказательства теоремы достаточно статической весовой функции, т.е. вес объекта зависит лишь от его размера. Функция из работы [11] проиллюстрирована ниже:

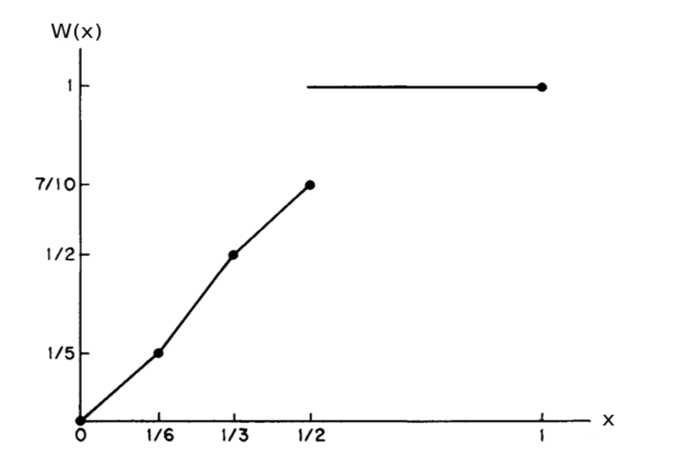


Рис. 1. Весовая функция
Fig. 1. Weighting function

Доказательство теоремы опирается на две Леммы:

Лемма 1: $\sum_{i=1}^{n} W(s(a_i)) \le \frac{17}{10} cost_{OPT}(\sigma)$.

Лемма 2: $\sum_{i=1}^{n} W(s(a_i)) \ge cost_{FF}(\sigma) - 2$.

Для доказательства Леммы 2 нужны два утверждения.

Утверждение 1: Если контейнер с грубостью $\alpha \le \frac{1}{2}$ заполнен объектами с весами $b_1, \dots, b_l$, и если $\sum_{i=1}^{l} b_i > 1 - \alpha$, то $\sum_{i=1}^{l} W(b_i) \ge 1$.

Утверждение 2: Если контейнер с грубостью $\alpha \le \frac{1}{2}$ заполнен объектами с весами $b_1 \ge \dots \ge b_l$ и $\sum_{i=1}^{l} W(b_i) = 1 - \beta$, то либо $m = 1$ и $b_1 < \frac{1}{2}$, либо $\sum_{i=1}^{l} b_i \le 1 - \alpha - \frac{5}{9}\beta$.

Далее, пусть для контейнеров $B'_1, \ldots, B'_m$ и только для них $\sum_{a_j \in B_i} W(s(a_j)) = 1 - \beta_i, \beta_i > 0$. Для любых $1 \le i < j \le m$ контейнер B'_j был создан позже B'_i. Так как в B'_m нет элемента размером $s(a_i) \ge \frac{1}{2}$, то грубость m-го контейнера $\alpha_m \le \frac{1}{2}$. По утверждению 2, $\alpha_i \ge \alpha_{i-1} + \frac{5}{9}\beta_{i-1}$, следовательно,

$\sum_{i=1}^{m-1} \beta_i \ge \frac{9}{5}\sum_{i=1}^{m-1}(\alpha_{i+1} - \alpha_i) = \frac{9}{5}(\alpha_m - \alpha_1) < \frac{9}{5} * \frac{1}{2} < 1.$

Из предыдущего неравенства и оценки $\beta_m \le 1$, имеем требуемое утверждение:

$$\sum_{i=1}^{n} W(s(a_{i)}) \ge cost_{FF}(\sigma) - 2$$

Отсюда получаем, что $R_{FF}^{\infty} \le 1.7$.

Для получения равенства достаточно привести пример сколь угодно большого набора σ с $cost_{FF}(\sigma) = \frac{17}{10} cost_{OPT}(\sigma)$ (рис. 2).

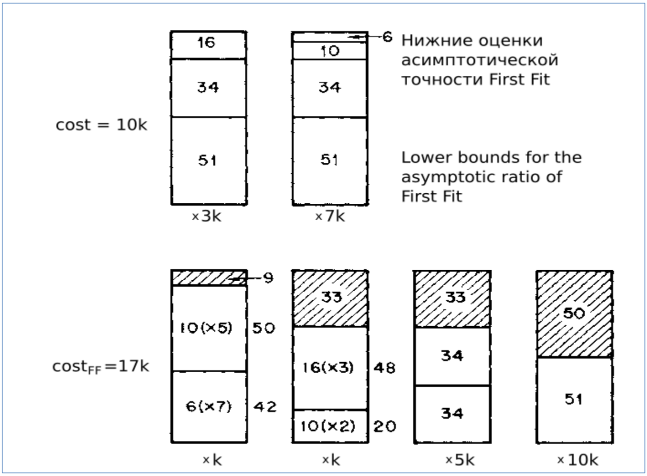


Рис. 2. Точная нижняя оценка First Fit
Fig. 2. Strict lower bounds for First Fit

Отметим, что первоначально в работе [12] в правой части леммы 2 вместо «-2» было «-3». Позднее в работе [13] «-2» было улучшено до «-1».

Заметим, что алгоритм First Fit создаёт новый контейнер тогда и только тогда, когда ни в одном из уже созданных контейнеров не достаточно места для размещения вновь прибывшего объекта. Скажем, что алгоритм принадлежит семейству **Any Fit**, если он удовлетворяет этому условию. В работе [10] доказано, что любой алгоритм AF из Any Fit имеет асимптотическую точность не лучше, чем First Fit.

Теорема 2 (Johnson, 1973) [10]. $R_{AF}^{\infty} \ge R_{FF}^{\infty}$ для любого Any Fit алгоритма AF.

Таким образом, при дальнейшем анализе можно либо ослабить требование онлайновости алгоритма, как, например, было сделано в работах [15] и [16], либо исследовать алгоритмы не из Any Fit, как сделал Яо в своей работе [14].

Скажем, что объект – типа A, если длина его лежит в интервале $(\frac{1}{2}, 1]$; типа B_1, если в $(\frac{2}{5}, \frac{1}{2}]$; типа B_2, если в $\left(\frac{1}{3}, \frac{2}{5}\right]$; типа X, если в $\left(0, \frac{1}{3}\right]$.

Алгоритм 1: Refined First Fit (RFF, Yao, 1981) [14]**.**

Пакуем объект каждого типа, кроме каждого шестого объекта типа B_2, в отдельный класс по алгоритму First Fit. Каждый шестой объект типа B_2 пакуем также по алгоритму First Fit в класс, где лежат все объекты типа A.

В [14] было также показано, что $R_{RFF}^{\infty} = \frac{5}{3}$. В дальнейшем был предложен целый ряд алгоритмов с уменьшенной асимптотической точностью; самый лучший из известных – алгоритм Harmonic++ из работы [17] с $R^{\infty} = 1.58889$. Однако,в среднем такие алгоритмы имеют асимптотику $W_A^n = \theta(n)$ и на практике редко используются.

В работе [14] было также показано, что для любого онлайнового алгоритма $R^{\infty} \geq \frac{3}{2}$. Было выбрано $0 < \varepsilon < 0.01$, и на вход алгоритму сначало поступало k объектов размером $\frac{1}{6} - 2\varepsilon$, затем – k объектов размером $\frac{1}{3} + \varepsilon$ и k объектов размером $\frac{1}{2} + \varepsilon$. Было доказано, что либо после упаковки k объектов, либо после упаковки $2k$ объектов, либо после упаковки всех $3k$ объектов число занятых контейнеров будет в $\frac{3}{2}$ раза больше числа контейнеров, занятых оптимальной упаковкой тех же объектов.

В работе [18] приведена лучшая оценка, чем в [14]: показано, что для любого онлайнового алгоритма $R^{\infty} \geq 1.536\ldots$ Лучшая из известных оценок $R^{\infty} \geq 1.540..$ была получена в работе [19] фон Влиетом.

3.2 Анализ в среднем

Часто нас интересует не эффективность работы алгоритма в худшем случае, а его среднее поведение. Известны алгоритмы упаковки при различных распределениях размеров упаковываемых объектов [20], однако в этом обзоре мы ограничимся равномерным распределением на отрезке $[0, v], v \leq 1$.

3.2.1 Размеры объектов имеют равномерное распределение на $[0, 1]$

Рассмотрим open-end случай, то есть случай, когда алгоритму неизвестно число объектов, которые он должен упаковать, до выпадения последнего из них.

В работе [23] была улучшена оценка модификации [22] теоремы из [21]:

Теорема 3 (Coffman, Shor, 1991) [23]. Если в квадрате случайно выбраны $2n$ точек, обозначим все точки на “+” или “-“ так, что у любой точки вероятности быть обозначенной плюсом и минусом равны. Пусть выбрано максимальное число пар “+ -“ так, что плюс в каждой паре правее минуса, и суммарная длина отрезков T, соединяющих плюс с минусом в каждой паре, минимальна. Тогда математическое ожидание $E(T) = O(\sqrt{n}\log^{\frac{3}{4}} n)$.

В работах [24] и [25] с помощью модификации этой теоремы была получена асимптотика для задачи Best Fit.

Теорема 4 (Shor, Leighton, 1989) [24, 25]. Математическое ожидание незаполненного пространства контейнеров при использовании алгоритма Best Fit W_{BF}^n=$\theta(\sqrt{n}\log^{\frac{3}{4}} n)$.

Приведём конструкцию для доказательства данной оценки.

Сначала каждому выпавшему объекту a_i, где $i = 1, \ldots, N$, ставится в соответствие точка в квадрате $[0,1] * [0,1]$: координата $x = s(a_i)$, координата $y = \frac{i}{N}$.

Плюсами помечаются объекты размером $s(a_i) \geq \frac{1}{2}$, а все остальные – минусами.

Затем весь квадрат отражается относительно прямой $x = \frac{1}{2}$ так, что плюсы и минусы попадают в образовавшийся прямоугольник $\left[0, \frac{1}{2}\right] * [0,1]$. Далее применяется теорема 3 для алгоритма MBF, или изменённого алгоритма Best Fit. Данный алгоритм отличается от Best Fit лишь тем, что контейнер закрывается после того, как в него попадает объект размером меньше $\frac{1}{2}$.

В работах [24] и [26] была оценена асимптотика алгоритма First Fit:

Теорема 5 (Coffman et al., 1991) [24, 26]. Математическое ожидание незаполненного пространства контейнеров при использовании алгоритма First Fit W_{BF}^n=$\theta(n^{\frac{2}{3}})$.

В работе [24] также была получена нижняя оценка W_A^n для любого онлайнового алгоритма:

Теорема 6 (Shor, 1986) [24]. Для любого онлайнового open-end алгоритма A математическое ожидание незаполненного пространства $W_A^n = \Omega(\sqrt{n \log n})$.

В работе [27] Шором был получен онлайновый open-end по Шору алгоритм с $W_A^n = O(\sqrt{n \log n})$.

Далее рассмотрим closed-end случай, когда алгоритм знает количество объектов, которые нужно упаковать до выпадения первого объекта a_1.

В работе [24] был предложен следующий алгоритм для упаковки n прямоугольников в режиме closed-end с $W_A^n = \theta(\sqrt{n})$:

1) для каждого из первых $\left[\frac{n}{2}\right]$ объектов создаём новый контейнер;

2) оставшиеся объекты пакуем алгоритмом Best Fit.

Таким образом, в closed-end случае нижняя оценка $W_{OPT}^n = \Omega(\sqrt{n})$ оказывается точной.

3.2.2 Размеры объектов имеют равномерное распределение на $[0, u], u < 1$

В [26] была получена нижняя оценка математического ожидания незаполненного пространства контейнеров:

Теорема 7 (Coffman et al, 1991) [26]. Пусть $u < 1$, размеры объектов выбираются попарно независимо согласно равномерному распределению на $[0, u]$ и A – онлайновый open-end алгоритм упаковки. Тогда математическое ожидание свободного пространства контейнеров $W_A^n = \Omega(\sqrt{n})$.

На данный момент неизвестны алгоритмы, на которых эта оценка достигается.

4. Многомерная задача Bin Packing

Многомерная задача Bin Packing в постановке упаковки в контейнеры Box Packing является естественным обобщением одномерной задачи Bin Packing (BPP). Как и BPP, она NP-трудна в сильном смысле, и для неё рассматриваются приближенные онлайновые алгоритмы.

4.1 Анализ в худшем случае

В работе [28] были получены нижние оценки для асимптотической точности онлайновых алгоритмов для многомерной задачи Bin Packing: для двухмерного случая $R_A^\infty \geq 1.802$, для трёхмерного случая $R_B^\infty \geq 1.974$ для любых онлайновых алгоритмов A и B.

Лучшая верхняя оценка для двумерной задачи была получена в [29], где предложен алгоритм с $R_B^\infty \leq 2.554$

Ксириком и фон Влиетом в [30] для d-мерной задачи был построен онлайновый алгоритм с асимптотической точностью $R_A^\infty = (\Pi_\infty)^d$, где $\Pi_\infty \approx 1.691$.

Важной для приложений характеристикой онлайнового алгоритма является открытое количество ящиков в каждый момент времени. Если в какой-то момент времени ящик закрыт, то после этого в него нельзя упаковать объект. Если в каждый момент времени число открытых ящиков не превышает некоторого наперед заданного числа k, то говорим, что алгоритм использует ограниченное пространство (bounded space).

В [31] Эпштейном и фон Сти был придуман алгоритм, использующий ограниченное пространство с такой же точностью $R_A^\infty = (\Pi_\infty)^d$. Кроме того, было доказано, что не существует алгоритма, использующего ограниченное пространство и имеющего лучшую асимптотическую точность.

4.2 Анализ в среднем

Ещё меньше результатов было получено при анализе в среднем. В работе [32] был предложен алгоритм Hash-Packing для d-мерной упаковки с точностью $W_A^n = O(n^{\frac{d+1}{d+2}})$ для длин сторон параллелипипедов, равномерно распределённых на $[0,1]$. Идея алгоритма заключается в нахождении среди поступающих d-мерных параллелипипедов групп параллелипипедов, достаточно плотно дополняющих друг друга до контейнера, и в отдельной упаковке групп параллелипипедов.

Алгоритм 2: Hash Packing (Chang, Wang, Kankanhalli, 1993) [32].

1. Выберем натуральное $m = m(n)$: $\exists N, \varepsilon, \delta > 0\ \forall n > N: n^{\varepsilon} < m < n^{\frac{1}{d}-\delta}$.
2. Скажем, что d-мерные прямоугольные параллелипипеды x_1 и x_2 лежат в одной группе, если для любых их соответствующих сторон $h_i(x_1)$ и $h_i(x_2)$ верно одно из условий:
   - на $[h_i(x_{1(\text{или }2)}), h_i(x_{2(\text{или }1)}))$ нет точек вида $\frac{j}{2m}, j = 1, \dots, 2m-1$;
   - на $[h_i(x_{1(\text{или }2)}), 1 - h_i(x_{2(\text{ или }1)}))$ имеется ровно одна точка вида $\frac{j}{2m},\ j = 1, \dots, 2m-1$.

3. У каждого прямоугольника в контейнере любой группы, в зависимости от длин его сторон, есть его место и, если в одном из контейнеров его группы его место свободно, то он на это место помещается. Иначе создаётся новый контейнер, куда и помещается параллелипипед на свое место.

Теорема 8 (Chang, Wang, Kankanhalli, 1993) [32]. Для предложенного алгоритма $W_A^n = \theta\left(\frac{n}{m}\right) + \theta(\sqrt{nm^d})$.

Первое слагаемое учитывает объём свободного пространства в контейнерах, заполненных всеми возможными (2^d) параллелипипедами, а второе – число не полностью заполненных контейнеров. Выбирая $m = n^{\frac{1}{d+2}}$, получаем $W_A^n = O(n^{\frac{d+1}{d+2}})$.

Алгоритм удовлетворяет условию open-end по Шору, для получения более сильного условия open-end с помощью алгоритма нужно упаковывать последовательно $2,\ 2^2, \dots,\ 2^k, \dots$ параллелипипедов, пока не получим символ останова.

5. Задача Strip Packing упаковки прямоугольников в полосу

Естественным обобщением задачи BP является задача Strip Packing упаковки прямоугольников в полубесконечную полосу. Задача впервые была исследована в [33], а в [34] Бэкером и Шварцем для применения алгоритмов из BP для задачи SP был предложен класс шельфовых алгоритмов, где шельф – такая часть полосы единичной ширины и ограниченной высоты, что прямоугольник не пересекается с границей шельфа, а лежит либо внутри, либо снаружи шельфа, и любая вертикальная прямая, пересекающая шельф, пересекает не более одного прямоугольника, лежащего внутри шельфа.

Алгоритм 3: Shelf(A,r). Пусть высота прямоугольника R, $h(R) \in (r^{k+1}, r^k]$, $r < 1$. Прямоугольник размещается в одном из шельфов высоты r^k. Для размещения прямоугольника в шельфах высоты r^k используем некоторую эвристику A для задачи BP в предположении, что множество шельфов данной высоты – это контейнеры, а ширина прямоугольника – вес упаковываемого объекта. При надобности алгоритм A создает новый шельф высоты r^k наверху текущей упаковки.

5.1 Анализ в худшем случае

Задача Strip Packing в случае, когда высоты всех прямоугольников одинаковы, эквивалентна задаче BP, стало быть, для онлайновой постановки задачи SP верна нижняя оценка из [19]: $R_A^\infty > 1.540$.

В [35] были предложены шельфрвые алгоритмы с асимптотичесской точностью R^∞, сколь угодно близкой к $\Pi_\infty (\Pi_\infty \approx 1.691)$, а также было показано, что для любого онлайнового шельфового алгоритма $R_A^\infty \geq \Pi_\infty$.

В [36] Ханом, Ивамом, Йе и Жангом был предложен алгоритм для задачи SP, имеющий асимптотическую точность $R_A^\infty \leq 1.588\ldots$ Для задач BP и SP неизвестен алгоритм с лучшей асимптотической точностью. Алгоритм разделяет прямоугольники на группы по ширине. Прямоугольники с шириной $w \leq \varepsilon_0$, упаковываются шельфовым алгоритмом $Shelf(Harmonic++, r)$. Пусть $\varepsilon_0 < t_1 <..< t_k = 1$, $c > h_{max}$, где h_{max}- максимальная высота прямоугольника. Прямоугольники, ширина которых $w \in (t_i, t_{i+1}]$, упаковывались в полоски высоты c и ширины t_{i+1}. Полоски одинаковой высоты, так как в данном случае задача SP эквивалентна задаче BP, размещались аналогом алгоритма Harmonic++ из [17] для BP.

5.2 Анализ в среднем

Задача исследовалась в предположении, что длины и высоты сторон-независимые в совокупности случайные велчины, равномерно распределённые на отрезке [0,1].

В 1993 году задача Strip Packing была исследована в среднем случае в [29] Коффманом и Шором. Было показано, что для любого шельфового алгоритма $W_{Shelf(A,r)}^n = \Omega(N^{\frac{2}{3}})$. Позднее в [38] были получены оценки точности онлайных шельфовых алгоритмов (в closed-end случае при наперед заданных значениях r), использующих алгоритмы First Fit и Best Fit для размещения прямоугольников внутри шельфов:

$$W_{Shelf(FF,r_{FF})}^n = O\left(N^{\frac{3}{4}}\right), W_{Shelf(BF,r_{BF})}^n = O\left(N^{\frac{2}{3}} \log^{\frac{1}{2}} N\right)$$

В работе [37] был предложен closed-end алгоритм с точностью $O\left(N^{\frac{2}{3}}\right)$, а 2011 году в [39] Кузюриным и Поспеловым был предложен open-end алгоритм с точностью $\theta\left(N^{\frac{2}{3}}\right)$. В работе [40] Трушниковым был предложен новый closed-end алгоритм, точность которого была исследована в работе [41], где было показано, что $W^n = O(N^{\frac{1}{2}} \log^{\frac{3}{2}} N)$. В [42] оценка была улучшена до

$$W^n = O\left(N^{\frac{1}{2}} \log N\right).$$

Значительное улучшение точности по сравнению с ранее известными алгоритмами обусловлено ограничением общей высоты разбиения на контейнеры и усовершенствованным способом упаковки прямоугольники в контейнеры.

Алгоритм 4 (Трушников, 2012) [40].

Пусть N – заранее известное число прямоугольников. Обозначим $d = \left\lfloor \frac{\sqrt{N}}{4} \right\rfloor, U = \frac{N}{4d}$. В основании полосы по левому краю выделяются прямоугольные области (контейнеры) высоты U, ширина i-ого контейнера равна $\frac{i}{d}$. Эти области образуют пирамиду. Рассмотрим вторую пирамиду, центрально симметричную данной относительно центра области, которая начинается от основания пирамиды, имеет высоту $(d+1)U,$ а ширина равна ширине полосы (см. рис. 3)

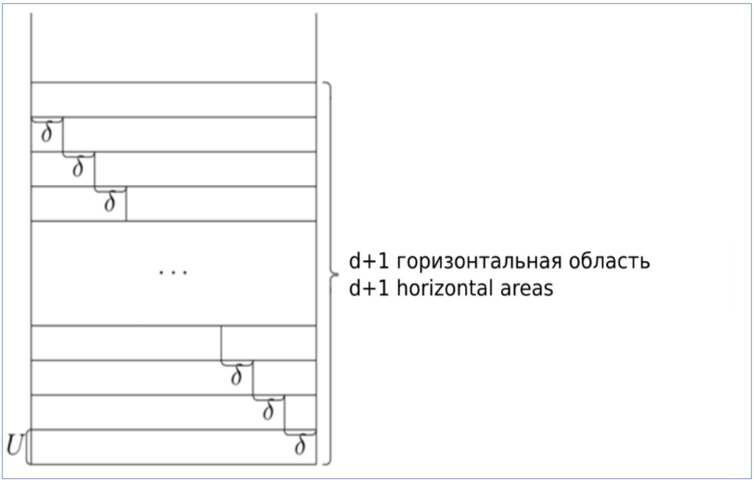


Рис. 3. Разбиение полосы на контейнеры
Fig. 3. Division of the strip into containers

Каждый четный прямоугольник размещается в одну пирамиду, а каждый нечетный – в другую. При упаковке каждого следующего прямоугольника шириной w:

- находим i: $\frac{i-1}{d} < w \le \frac{i}{d}$;
- ищем $\min j : i \le j \le d$ такое, что прямоугольник помещается в контейнер шириной $\frac{j}{d}$;
- если такое j существует, кладём прямоугольник на верхнюю грань верхнего прямоугольника в контейнере шириной $\frac{j}{d}$;
- иначе называем прямоугольник выпавшим и кладём его наверх текущей упаковки.

Принципиальным отличием данного алгоритма от его предшественников является то, что число контейнеров $(2d)$ – зависит от числа прямоугольников, но не зависит от самих прямоугольников.

Для задачи Strip Packing известна лишь очевидная нижняя оценка, верная как для online, так и для offline алгоритмов: $W^n = \Omega(N^{\frac{1}{2}})$.

6. Задача Multiple Strip Packing упаковки прямоугольников в несколько полос единичной ширины

Для практических приложений, например, для задач оптимизации в распределённых вычислительных системах [43] полезно, перейти от рассмотрения задачи об упаковке прямоугольников в одну полосу к рассмотрению задачи об упаковке прямоугольников в несколько полос (MSP). В данной задаче продуктивна идея об упаковке некоторого объекта в наименее заполненную полосу, где объектами могут быть шельфы для шельфовых алгоритмов или выпавшие прямоугольники для алгоритмов из [40].

6.1 Анализ в худшем случае

Рассмотрим шельфовый алгоритм упаковки прямоугольников в несколько полос, предложенный в работе [44].

Алгоритм 5 (Йе, Хан, Жанг, 2011) [44]:

- выбираем шельф, в который кладётся прямоугольник согласно эвристике $Shelf(A, r)$;
- если такого шельфа не существует, создаем новый шельф наверху текущей упаковки в полосе, заполненной не выше любой другой полосы.

Аналогично [35], можно получить семейство шельфовых алгоритмов с асимптотической точностью R^∞, сколь угодно близкой к $\Pi_\infty (\Pi_\infty \approx 1.691)$.

Размещая же каждый контейнер согласно алгоритму из работы [36] в наименее заполненную по высоте полосу, можно получить для случая MSP оценку из [36]: $R_A^\infty \leq 1.588...$

6.2 Анализ в среднем

Алгоритм 4 может быть обобщён на случай нескольких полос упаковкой выпавших прямоугольников в наименее высоко заполненный контейнер.

В работе [42] алгоритм был обобщен на случай, когда число полос для упаковки зависит от числа прямоугольников. Был предложен алгоритм упаковки N прямоугольников в k полос, $k \leq \sqrt{N}$. Для него была доказана следующая теорема:

Теорема 9 (Лазарев, Кузюрин, 2017) [42]. При упаковке N прямоугоьников в k полос, $k \leq \sqrt{N}$, алгоритм обеспечивает площадь незаполненного пространства $W^n = O(N^{\frac{1}{2}} \log\, N)$.

Для задачи Multiple Strip Packing, как и для задачи SP, известна лишь очевидная нижняя оценка, верная как для online, так и для offline алгоритмов: $W^n = \Omega(N^{\frac{1}{2}})$.

7. Задача об упаковке прямоугольников в полосы различной ширины. Анализ в худшем случае

Обобщением задачи MSP является задача об упаковке прямоугольников в полосы различной ширины.

Дан набор полубесконечных полос $C = \{C_1, \dots, C_m\}$, w_i – ширина i-ой полосы. В эти полосы требуется упаковать без вращений и пересечений n открытых прямоугольников $T = \{T_1, \dots, T_n\}$, минимизируя при этом высоту упаковки.

Отличие от задачи MSP заключается лишь в том, что ширина у полос различная.

В [45, 46] задача впервые была рассмотрена Жуком, и был предложен алгоритм, распределяющий прямоугольники в полосы в режиме онлайн, а в полосах использующий оффлайновую эвристику для упаковки с асимптотической точностью $R_A^\infty < 10$. В [47] для $\forall\, r \in (0,1)$ был предложен полностью онлайновый алгоритм A_r с асимптотической точностью $R_{A(r)}^\infty \leq \frac{8}{r}$. В [48] Жук доказал, что для любого онлайнового алгоритма $R_A^\infty \geq e$, $e = 2{,}7182\ldots$. В работе [49] был предложен онлайновый алгоритм с асимптотической точностью $R_A^\infty < \frac{2e}{r}, \forall\, r \in (0,1)$.

Предположим, что $w_1 \geq \cdots \geq w_m$. Пусть $R -$ прямоугольник с шириной $w(R)$. Скажем, что $last(R) = max\{k : w_k \geq w(R)\}$. Разобьем все выпавшие прямоугольники $\{T\}$ на m множеств $M_1, \dots, M_m : R \in M_i \Leftrightarrow last(R) = i$.

Если Q – множество прямоугольников, то за $S(Q)$ обозначим суммарную площадь прямоугольников из множества Q.

Определим $h(T) = \sup_k \frac{\sum_{i=1}^{k} S(M_i)}{\sum_{i=1}^{k} w_i}$. Нетрудно показать, что высота оптимального распределения $H_o \geq h$.

За $y_i(T)$ обозначим суммарную площадь прямоугольников, попавших в i-ую полосу после выпадения множества T прямоугольников.

Алгоритм 6: A_r (Жук, 2012) [49].

При добавлении очередного прямоугольника R:

- вычисляем $h = h(T + \{R\})$;
- определяем номер полосы k: $k = \max\left\{i: w(R) \leq w_i, \frac{y_i}{w_i} \leq eh\right\}$ (было доказано, что такая полоса всегда существует);
- для размещения внутри полосы используется шельфовый алгоритм $Shelf(r), r < 1$, разбивающий полосу на слои высотой $r^z, z \in \mathbb{Z}$; алгоритм кладёт прямоугольник R высотой $h(R)$ в слой высотой r^x ($r^{x-1} < h(R) \leq r^x$), а для упаковки в слой используется эвристика First Fit.

Для получения асимптотической точности R_A^∞ используется следующая лемма.

Лемма. При упаковке прямоугольников высотой не больше h_{max} суммарной площадью S алгоритмом $Shelf(r)$ в полосу ширины w высота упаковки

$$H_{Shelf(r)} \leq \frac{2S}{rw} + h_{max}(1 + \frac{1}{r(1-r)})$$

Для доказазательства леммы нужно рассмотреть слои, заполненные менее чем на $\frac{w}{2}$ по ширине, слои, заполненные более чем наполовину по ширине, и последний слой.

Из леммы получаем, что для алгоритма A_r мультипликативная асимптотическая точность $R_A^\infty \leq 2e$.

8. Заключение

Классическая задача Bin Packing очень хорошо изучена: при анализе в худшем случае показано, что для любого онлайнового алгоритма $R^\infty \geq 1.540$, и был предложен алгоритм с $R^\infty \leq 1.589$. При анализе в среднем как в случае open-end, так и в случае closed-end были предложены алгоритмы, у которых математическое ожидание площади незаполненного пространства контейнеров W^n имеет неулучшаемую асимптотику.

Однако многие обобщения этой задачи активно изучаются в настоящее время, и многие результаты еще предстоит получить. В частности, в многомерном обобщении Multidimensional Bin Packing задачи Bin Packing при анализе в среднем неизвестны нижние оценки, кроме очевидных $W^n = \Omega(\sqrt{n})$.

В задачах Strip Packing и Multiple Strip Packing при анализе в худшем случае был предложен алгоритм с $R^\infty \leq 1.589$, а при анализе в среднем в closed-end случае, был предложен алгоритм с $W^n = O(\sqrt{n}\ln n)$, однако при анализе в постановке open-end неизвестна нижняя оценка W^n, лучшая $W^n = \Omega\left(\sqrt{n}\right)$, и неизвестен алгоритм с $W^n = o(n^{2/3})$.

Задача упаковки прямоугольником в полосы различной ширины активно исследовалась в худшем случае, однако в среднем задача еще не была проанализирована.

Список литературы

[1]. Массобрио Р., Несмачнов С., Черных А., Аветисян А., Радченко Г. Применение облачных вычислений для анализа данных большого объёма в умных городах. Труды ИСП РАН, том 28, вып. 6, 2016 г., стр. 121-140, DOI: 10.15514/ISPRAS-2016-28(6)-9

[2]. Аничкин А.С., Семенов В.А. Математическая формализация задач проектного планирования в расширенной постановке. Труды ИСП РАН,том 29, вып. 2, 2017 г., стр. 231-256. DOI: 10.15514/ISPRAS-2017-29(2)-9

[3]. Зеленова С.А., Зеленов С.В. Критерий существования бесконфликтного расписания для системы строго периодических задач. Труды ИСП РАН, том 29, вып. 6, 2017 г., стр. 183-202. DOI: 10.15514/ISPRAS-2017-29(6)-10

[4]. Ghalam L., Grosu D. A Parallel Approximation Algorithm for Scheduling Identical Machines. In IEEE International Parallel and Distributed Processing Symposium Workshops, 2017, pp. 619-628

[5]. Sheikhalishahi M., Wallace R. M., Grandinetti L., Vazquez-Poletti J. L., Guerriero F. A multi-dimensional job scheduling. Future Generation Computer Systems, vol. 54, 2016, pp. 123-131

[6]. Tchernykh A., Schwiegelshohn U., Yahyapour R., Kuzjurin N. On-line hierarchical job scheduling on grids with admissible allocation. Journal of Scheduling, 2010, vol. 13, issue 5, pp. 545-552

[7]. Tshernykh A., Ramirez J.M., Avetisyan A., Kuzjurin N., Grushin D., Zhuk S. Two-Level Job-Scheduling strategies for a Computational Grid. Lecture Notes in Computer Science book series, vol. 3911, pp. 774-781

[8]. Cohil B., Shah S., Goleshha Y., Patel D. A Comparative Analysis of Virtual Machine Placement Techniques in the Cloud Environment. International Journal of Computer Applications, vol. 156, no. 14, 2016, pp. 12-18

[9]. Garey M.R., Johnson D.S. Computers and intractability: a guide to the theory of NP-completeness. Freeman: San Francisco, 1979, 338 p.

[10]. Johnson D.S. Near-optimal Bin Packing Algorithms. PhD Thesis, Massachusetts Institute of Technology, Department of Mathematics, Cambridge, 1973. 401 p.

[11]. Johnson D.S., Demers A., Ullman J.D., Garey M.R., Graham R.L. Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. SIAM Journal on computing, vol. 3, issue 4, 1974, pp. 299- 325

[12]. Garey M.R. Graham R.L., Ullman J.D., Worst-case analysis of memory allocation algorithms. Proceedings of the fourth annual ACM symposium on theory of computing. 1972, pp. 143-150

[13]. Garey M.R., Graham R.L., Johnson D.S., Yao A.C. Resource constrained scheduling as generalized bin packing. Journal of Combinatorial Theory, Series A, vol. 21, issue 3, 1976, pp. 257-298

[14]. Yao A.C. New Algorithms for Bin Packing. Journal of the ACM, vol. 27, issue 2, 1981, pp. 207-227

[15]. Gambosi G., Postiglione a., Talamo M.M. New algorithms for online Bin Packing. In Proceedings of the First Italian Conference on Algorithms and Complexity, 1990, pp. 44-59

[16]. Ivcović Z. and Lloyd E. Fully dynamic algorithms for Bin Packing: Being (mostly) myopic helps. Lecture Notes in Computer Science, vol. 726, pp. 224-235

[17]. Seiden S.S. On the Online Bin Packing Problem. Lecture Notes in computer science, vol. 2076, 2002, pp. 207-227

[18]. Brown J.D. A lower Bound for On-Line One Dimensional Bin Packing Algorithms. Technical Report R-864, coordinated Science laboratory, University of Illinois, Urbana, IL, 1979.

[19]. Vliet A. An improved lower bound for on-line bin packing algorithms. Information Processing Letters, vol. 43, issue 5, 1992, pp. 277-284

[20]. Breitgand D., Epstein A. Improving consolidations of virtual machines with risk-aware Bandwidth oversubscription in compute clouds. In Proceedings of the IEEE INFCOM, 2012, pp. 2861-2865

[21]. Ajtai M., Komlós J., Tusnadi G. On Optimal Matchings. Combinatorica, vol. 4, issue 4, 1984, pp. 259-264

[22]. Karp R.M., Luby M., Marchetti-Spaccamela A. A probabilistic analysis of multidimensional bin packing problem. In Proceedings of the sixteen annual ACM symposium on theory of computing, 1984, pp.289-298

[23]. Coffman E.G., Shor P.W. A Simple Proof of the sqrt(n log3/4 n) Upright Matching Bound. SIAM Journal on Discrete Mathematics, vol. 4, issue 1, 1991, pp. 48-57

[24]. Shor P.W. The average-case analysis of some on-line algorithms for bin packing. Combinatorica, vol. 6, issue 4, 1986, pp. 179-200

[25]. Leighton F.T., Shor P. Tight bonds for minimax grid matching, with application to the average-case analysis of algorithms. In Proceedings of the eighteenth Annual ACM symposium on theory of computing, 1986 , pp. 91-103

[26]. Coffman E.G., Courcoubetis C., Garey M.R., Johnson D.S., McGeoch L.A., Shor P.W., Weber R. and Yannakakis M. Fundamental discrepancies between average-case analysis under discrete and continuous distributions: a bin packing study. In Proceedings of the Twenty-first Annual ACM symposium on theory of computing, 1991, pp. 230-240

[27]. Shor P.W. How to pack better than Best Fit: tight bounds for average-case online Bin Packing. In Proceedings 32nd of the Annual Symposium of foundations of Computer Science, 1991, pp. 752-759

[28]. Galambos G., A. van Vliet. Lower bounds for 1-, 2-, and 3- dimensional on-line bin packing algorithms. Computing, vol. 52, issue 3, 1994, pp. 281-297

[29]. Han X., Chin F.Y.L., Ting H.-F., Zhang G., Zhang Y. A new upper bound 2.5545 on 2D Online Bin Packing. ACM Transactions on algorithms, vol. 7, issue 4, 2011, article No. 50

[30]. Csirik J., A. van Vliet. An on-line algorithm for multidimensional bin packing. Operation Research Letters, vol. 13, issue 3, 1993, pp. 149-158
[31]. Epstein l., R. van Stee. Optimal Online Algorithms for Multidimensional Packing Problems. In Proceedings of the Fifteenth Annual ACM-CIAM Symposium on Discrete algorithms, 2004, pp. 214-223
[32]. Chang E-C, Wang W., Kankanhalli M.S. Multidimensional on-line bin-packing: An algorithm and it's average-case analysis. Information Processing Letters, vol. 48, issue 3, 1993, pp. 121-125
[33]. Baker B.S., Coffman E.G., Rivest R.L. Orthogonal Packings in Two Dimensions. SIAM Journal on Computing, vol. 9, issue 4, 1980, pp. 846-855
[34]. Baker B.S., Schwarz J.S. Shelf algorithms for two-dimensional packing problems. SIAM Journal on Computing, vol. 12, issue 3, 1983, pp. 508-525
[35]. Csirik J., Woeginger G.J. Shelf algorithms for on-line strip packing. Information Processing Letters, vol. 63, issue 4, 1997, pp. 171-175
[36]. Han X., Iwama K., Ye d., Zhang G. Strip Packing vs Bin Packing. Lecture Notes in Computer Science, vol. 4508, 2007, pp. 358-367
[37]. Coffman E.G., Shor P.W. Packing in two dimensions: Asymptotic average-case analysis of algorithms. Algorithmica, vol. 9, issue 3, 1993, pp. 253-277
[38]. Кузюрин Н.Н., Поспелов А.И. Вероятностный анализ различых шельфовых алгоритмов упаковки прямоугольников в полосу. Труды ИСП РАН, том 12, 2007 г., стр. 17-26
[39]. Кузюрин Н.Н., Поспелов А.И. Вероятностный анализ нового класса алгоритмов упаковки прямоугольников в полосу. Ж. вычисл. матем. и матем. физ., том 51, no. 10, 2011, стр. 1931-1936
[40]. Трушников М.А. Об одной задаче Коффмана-Шора, связанной с упаковкой прямоугольников в полосу. Труды ИСП РАН, том 22, 2012 г., стр. 456-462, DOI: 10.15514/ISPRAS-2012-22-24
[41]. Трушников М.А. Вероятностный анализ нового алгоритма упаковки прямоугольников в полосу. Труды ИСП РАН, том 24, 2013 г., стр. 457-468, DOI: 10.15514/ISPRAS-2013-24-21
[42]. Лазарев Д.О., Кузюрин Н.Н. Алгоритм упаковки прямоугольников в несколько полос и анализ его точности в среднем. Труды ИСП РАН, том 29, вып. 6, 2017 г., стр. 221-228, DOI: 10.15514/ISPRAS-2017-29(6)-13
[43]. Кузюрин Н.Н., Грушин Д.А., Фомин А. Проблемы двумерной упаковки и задачи оптимизации в распределённых вычислительных системах. Труды ИСП РАН, том 26, вып. 1, 2014 г., стр. 483-502, DOI: 10.15514/ISPRAS-2014-26(1)-21
[44]. Ye D., Han X., Zhang G. Online multiple-strip packing. Theoretical Computer Science, vol. 412, issue 3, 2011, pp. 233-239
[45]. Zhuk S.N. Approximation algorithms for packing rectangles into several strips. Discrete Mathematics and Applications, vol. 16, issue 1, 2006, pp. 73-85
[46]. Жук С.Н. Анализ некоторых эвристик упаковки прямоугольников в несколько полос. Труды ИСП РАН, том 6, 2004 г., стр. 13-26
[47]. Жук С.Н. Онлайновый алгоритм упаковки прямоугольников в несколько полос с гарантированными оценками точности. Труды ИСП РАН, том 12, 2007 г., стр. 7-16
[48]. Zhuk S.N. On-line algorithms for packing rectangles into several strips. Discrete Mathematics and Applications, vol. 17, issue 5, 2007, pp. 517-531

Жук С.Н. О построении расписаний выполнения параллельных задач на группах кластеров с различной производительностью. Труды ИСП РАН, том 23, 2012 г., стр. 447-454, DOI: 10.15514/ISPRAS-2012-23-27

On on-line algorithms for Bin, Strip and Box Packing, and their worst- and average-case analysis

[1,2] *D.O. Lazarev <dennis810@mail.ru>*
[1,2] *N.N. Kuzjurin <nnkuz@ispras.ru>*
[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Moscow Institute of Physics and Technology, Dolgoprudnyj, Institutskij alley, Moscow region, 141700, Russia*

Abstract. In this survey, online algorithms for such packing problems as Bin Packing, Strip Packing and their generalizations, such as Multidimensional Bin Packing, Multiple Strip Packing and packing into strips of different width were considered. For the latter problem only worst-case analysis was described, for all other problems, both worst-case and average case (probabilistic analysis) asymptotical ratios were considered. Both lower and upper bonds were described. Basic algorithms and methods for their analysis were considered. For worst-case analysis of the Bin Packing Problem it was shown that for any online algorithm $\boldsymbol{R}^{\infty} \geq \mathbf{1.540}$, and an algorithm with $\boldsymbol{R}^{\infty} \leq \mathbf{1.589}$ was obtained. Both approaches for analyzing the algorithm and obtaining the lower bonds were discussed. Also it was shown that First Fit algorithm for Bin Packing has asymptotical competitive ratio of $\frac{\mathbf{17}}{\mathbf{10}}$. For average case analysis in the case when object's sizes have a uniform distribution on $[\mathbf{0},\mathbf{1}]$ in open-end analysis a construction for obtaining both lower bound of $\boldsymbol{W^n} = \boldsymbol{\Omega}(\sqrt{\boldsymbol{n}\,\boldsymbol{ln}\,\boldsymbol{n}})$ and algorithm with $\boldsymbol{W^n} = \boldsymbol{\theta}(\sqrt{\boldsymbol{n}\,\boldsymbol{ln}\,\boldsymbol{n}})$ was shown. In the case of closed-end analysis an algorithm with $\boldsymbol{W^n} = \boldsymbol{\theta}(\sqrt{\boldsymbol{n}})$ was described. For Multidimensional Bin Packing with $\boldsymbol{d}$ dimensions an algorithm with $\boldsymbol{R}^{\infty} = (\Pi_{\infty})^{\boldsymbol{d}}$, where $\Pi_{\infty} \approx \mathbf{1.691}$, was obtained. For average case analysis an algorithm with $\boldsymbol{W_A^n} = \boldsymbol{O}(\boldsymbol{n}^{\frac{\boldsymbol{d}+1}{\boldsymbol{d}+2}})$ was shown. For worst-case analysis of Strip Packing Problem and Multiple Strip Packing Problem algorithms with $\boldsymbol{R}^{\infty} \leq \mathbf{1.589}$ were also obtained. For the closed-end average case analysis algorithms with $\boldsymbol{W^n} = \boldsymbol{\theta}(\sqrt{\boldsymbol{n}}\,\boldsymbol{ln}\,\boldsymbol{n})$ were described. For the open-end average case analysis of Strip Packing Problem an algorithm with $\boldsymbol{W^n} = \boldsymbol{O}\left(\boldsymbol{n}^{\frac{2}{3}}\right)$ was considered. For generalization of Multiple Strip Packing Problem, where strips have different widths, an online algorithm with $\boldsymbol{R}^{\infty} \leq \frac{\mathbf{2}\boldsymbol{e}}{\boldsymbol{r}}$ for any $\boldsymbol{r} < \mathbf{1}$, where $\boldsymbol{e} = \mathbf{2.718}\ldots$, was described.

Keywords: Bin Packing; Multidimensional Bin Packing; Strip Packing; Multiple Strip Packing; Packing in Strips of different width; probabilistic analysis; worst-case analysis.

DOI: 10.15514/ISPRAS-2018-30(4)-14

For citation: Lazarev D.O., Kuzjurin N.N. On on-line algorithms for Bin, Strip and Box Packing, and their worst- and average-case analysis. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 4, 2018. pp. 209-230 (in Russian). DOI: 10.15514/ISPRAS-2018-30(4)-14

References

[1]. Massobrio R., Nesmachnow S., Tchernykh A., Avetisyan A., Radchenko G. Towards a Cloud Computing Paradigm for Big Data Analysis in Smart Cities. Trudy ISP RAN/Proc.ISP RAS, vol. 28, issue 6, 2016. pp. 121-140 (in Russian). DOI: 10.15514/ISPRAS-2016-28(6)-9

[2]. Anichkin A.S., Semenov V.A. Mathematical formalization of project scheduling problems. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 2, 2017. pp. 231-256 (in Russian). DOI: 10.15514/ISPRAS-2017-29(2)-9

[3]. Zelenova S.A., Zelenov S.V. Non-conflict scheduling criterion for strict periodic tasks. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 6, 2017. pp. 183-202 (in Russian). DOI: 10.15514/ISPRAS-2017-29(6)-10

[4]. Ghalam L., Grosu D. A Parallel Approximation Algorithm for Scheduling Identical Machines. In IEEE International Parallel and Distributed Processing Symposium Workshops, 2017, pp. 619-628

[5]. Sheikhalishahi M., Wallace R. M., Grandinetti L., Vazquez-Poletti J. L., Guerriero F. A multi-dimensional job scheduling. Future Generation Computer Systems, vol. 54, 2016, pp. 123-131

[6]. Tchernykh A., Schwiegelshohn U., Yahyapour R., Kuzjurin N. On-line hierarchical job scheduling on grids with admissible allocation. Journal of Scheduling, 2010, vol. 13, issue 5, pp. 545-552

[7]. Tshernykh A., Ramirez J.M., Avetisyan A., Kuzjurin N., Grushin D., Zhuk S. Two-Level Job-Scheduling strategies for a Computational Grid. Lecture Notes in Computer Science book series, vol. 3911, pp. 774-781

[8]. Cohil B., Shah S., Goleshha Y., Patel D. A Comparative Analysis of Virtual Machine Placement Techniques in the Cloud Environment. International Journal of Computer Applications, vol. 156, no. 14, 2016, pp. 12-18

[9]. Garey M.R., Johnson D.S. Computers and intractability: a guide to the theory of NP-completeness. Freeman: San Francisco, 1979, 338 p.

[10]. Johnson D.S. Near-optimal Bin Packing Algorithms. PhD Thesis, Massachusetts Institute of Technology, Department of Mathematics, Cambridge, 1973. 401 p.

[11]. Johnson D.S., Demers A., Ullman J.D., Garey M.R., Graham R.L. Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. SIAM Journal on computing, vol. 3, issue 4, 1974, pp. 299- 325

[12]. Garey M.R. Graham R.L., Ullman J.D., Worst-case analysis of memory allocation algorithms. Proceedings of the fourth annual ACM symposium on theory of computing. 1972, pp. 143-150

[13]. Garey M.R., Graham R.L., Johnson D.S., Yao A.C. Resource constrained scheduling as generalized bin packing. Journal of Combinatorial Theory, Series A, vol. 21, issue 3, 1976, pp. 257-298

[14]. Yao A.C. New Algorithms for Bin Packing. Journal of the ACM, vol. 27, issue 2, 1981, pp. 207-227

[15]. Gambosi G., Postiglione a., Talamo M.M. New algorithms for online Bin Packing. In Proceedings of the First Italian Conference on Algorithms and Complexity, 1990, pp. 44-59

[16]. Ivcović Z. and Lloyd E. Fully dynamic algorithms for Bin Packing: Being (mostly) myopic helps. Lecture Notes in Computer Science, vol. 726, pp. 224-235

[17]. Seiden S.S. On the Online Bin Packing Problem. Lecture Notes in computer science, vol. 2076, 2002, pp. 207-227

[18]. Brown J.D. A lower Bound for On-Line One Dimensional Bin Packing Algorithms. Technical Report R-864, coordinated Science laboratory, University of Illinois, Urbana, IL, 1979.

[19]. Vliet A. An improved lower bound for on-line bin packing algorithms. Information Processing Letters, vol. 43, issue 5, 1992, pp. 277-284

[20]. Breitgand D., Epstein A. Improving consolidations of virtual machines with risk-aware Bandwidth oversubscription in compute clouds. In Proceedings of the IEEE INFCOM, 2012, pp. 2861-2865

[21]. Ajtai M., Komlós J., Tusnadi G. On Optimal Matchings. Combinatorica, vol. 4, issue 4, 1984, pp. 259-264

[22]. Karp R.M., Luby M., Marchetti-Spaccamela A. A probabilistic analysis of multidimensional bin packing problem. In Proceedings of the sixteen annual ACM symposium on theory of computing, 1984, pp.289-298

[23]. Coffman E.G., Shor P.W. A Simple Proof of the sqrt(n log3/4 n) Upright Matching Bound. SIAM Journal on Discrete Mathematics, vol. 4, issue 1, 1991, pp. 48-57

[24]. Shor P.W. The average-case analysis of some on-line algorithms for bin packing. Combinatorica, vol. 6, issue 4, 1986, pp. 179-200

[25]. Leighton F.T., Shor P. Tight bonds for minimax grid matching, with application to the average-case analysis of algorithms. In Proceedings of the eighteenth Annual ACM symposium on theory of computing, 1986 , pp. 91-103

[26]. Coffman E.G., Courcoubetis C., Garey M.R., Johnson D.S., McGeoch L.A., Shor P.W., Weber R. and Yannakakis M. Fundamental discrepancies between average-case analysis under discrete and continuous distributions: a bin packing study. In Proceedings of the Twenty-first Annual ACM symposium on theory of computing, 1991, pp. 230-240

[27]. Shor P.W. How to pack better than Best Fit: tight bounds for average-case online Bin Packing. In Proceedings 32nd of the Annual Symposium of foundations of Computer Science, 1991, pp. 752-759

[28]. Galambos G., A. van Vliet. Lower bounds for 1-, 2-, and 3- dimensional on-line bin packing algorithms. Computing, vol. 52, issue 3, 1994, pp. 281-297

[29]. Han X., Chin F.Y.L., Ting H.-F., Zhang G., Zhang Y. A new upper bound 2.5545 on 2D Online Bin Packing. ACM Transactions on algorithms, vol. 7, issue 4, 2011, article No. 50

[30]. Csirik J., A. van Vliet. An on-line algorithm for multidimensional bin packing. Operation Research Letters, vol. 13, issue 3, 1993, pp. 149-158

[31]. Epstein l., R. van Stee. Optimal Online Algorithms for Multidimensional Packing Problems. In Proceedings of the Fifteenth Annual ACM-CIAM Symposium on Discrete algorithms, 2004, pp. 214-223

[32]. Chang E-C, Wang W., Kankanhalli M.S. Multidimensional on-line bin-packing: An algorithm and it’s average-case analysis. Information Processing Letters, vol. 48, issue 3, 1993, pp. 121-125

[33]. Baker B.S., Coffman E.G., Rivest R.L. Orthogonal Packings in Two Dimensions. SIAM Journal on Computing, vol. 9, issue 4, 1980, pp. 846-855
[34]. Baker B.S., Schwarz J.S. Shelf algorithms for two-dimensional packing problems. SIAM Journal on Computing, vol. 12, issue 3, 1983, pp. 508-525
[35]. Csirik J., Woeginger G.J. Shelf algorithms for on-line strip packing. Information Processing Letters, vol. 63, issue 4, 1997, pp. 171-175
[36]. Han X., Iwama K., Ye d., Zhang G. Strip Packing vs Bin Packing. Lecture Notes in Computer Science, vol. 4508, 2007, pp. 358-367
[37]. Coffman E.G., Shor P.W. Packing in two dimensions: Asymptotic average-case analysis of algorithms. Algorithmica, vol. 9, issue 3, 1993, pp. 253-277
[38]. Kuzjurin N.N., Pospelov A.I. Probabilistic analysis of different shelf algorithms for packing rectangles into a strip. Trudy ISP RAN/Proc. ISP RAS, vol. 12, 2007, pp. 17-26 (in Russian)
[39]. Kuzyurin N.N., Pospelov A.I. Probabilistic analysis of a new class of strip packing algorithms. Comput. Math. and Math. Phys., vol. 51, issue 10, 2011, article no. 1817
[40]. Trushnikov M.A. On one problem of Koffman-Shor connected to strip packing problem. Trudy ISP RAN/Proc. ISP RAS, vol. 22, 2012, pp. 456-462 (in Russian). DOI: 10.15514/ISPRAS-2012-22-24
[41]. Trushnikov M.A. Probabilistic analysis of a new strip packing algorithm. Trudy ISP RAN/Proc. ISP RAS, vol. 24, 2013, str. 457-468 (in Russian). DOI: 10.15514/ISPRAS-2013-24-21
[42]. Lazarev D.O., Kuzyrin N.N. An algorithm for Multiple Strip Package and its average case evaluation. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 6, 2017. pp. 221-228 (in Russian). DOI: 10.15514/ISPRAS-2017-29(6)-13
[43]. Kuzjurin N.N., Grushin D.A., Fomin A. Two-dimensional packing problems and optimization in distributed computing systems. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 1, 2014, pp. 483-502 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-21
[44]. Ye D., Han X., Zhang G. Online multiple-strip packing. Theoretical Computer Science, vol. 412, issue 3, 2011, pp. 233-239
[45]. Zhuk S.N. Approximation algorithms for packing rectangles into several strips. Discrete Mathematics and Applications, vol. 16, issue 1, 2006, pp. 73-85
[46]. Zhuk S.N. Analysis of some heuristics of packing rectangles into several strips. Trudy ISP RAN/Proc. ISP RAS, vol. 6, 2004, pp. 13-26 (in Russian)
[47]. Zhuk S.N. Online algorithm for packing rectangles into several strips with guaranteed accuracy estimates. Trudy ISP RAN/Proc. ISP RAS, vol. 12, 2007, pp. 7-16 (in Russian)
[48]. Zhuk S.N. On-line algorithms for packing rectangles into several strips. Discrete Mathematics and Applications, vol. 17, issue 5, 2007, pp. 517-531
[49]. Zhuk S.N. On-line algorithm for scheduling parallel tasks on a group of related clusters. Trudy ISP RAN/Proc. ISP RAS, vol. 23, 2012, pp. 447-454 (in Russian). DOI: 10.15514/ISPRAS-2012-23-27