ИСП Институт Системного Программирования
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 29, выпуск 4

Volume 29, issue 4

Москва 2017

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:

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

Automated Type Contracts Generation in Ruby

[1,2] N. Y. Viuginov <viuginov.nickolay@gmail.com>
[2] V. S. Fondaratov <fondarat@gmail.com>
[1] St. Petersburg State University,
13B Universitetskaya Emb., St. Petersburg, 199034, Russia
[2] JetBrains,
7-9-11 Universitetskaya Emb., St. Petersburg, 199034, Russia

Abstract. Elegant syntax of the Ruby language pays back when it comes to finding bugs in large codebases. Static analysis is hindered by specific capabilities of Ruby, such as defining methods dynamically and evaluating string expressions. Even in dynamically typed languages, type information is very useful as it ensures better type safety and more reliable checking whether the called method is defined for the object or whether the arguments of the correct types are passed to it. One may annotate the code with YARD (Ruby documentation tool) to declare the input and output types of methods or even declare methods that are added dynamically. These annotations improve the capabilities of tooling such as code completion. This paper reports a new approach to type annotations generation. We trace direct method calls while the program is running, evaluate types of input and output variables and use this information to derive implicit type contracts. Each method or function is associated with a finite-state automaton consisting of all variants of typed signatures for this method. An effective compression technique is applied to the automaton to reduce the cost of storage and allows to display the collected information in a human-readable form. The exhaustiveness of the contract defined by the generated automaton depends on the diversity of the traced method usages. Therefore, it is also important to be able to merge all the automatons received from users into one, which is further covered in this paper.

Keywords: Ruby; dynamically typed languages; Ruby VM; YARV; method signature; type inference; static code analysis

DOI: 10.15514/ISPRAS-2017-29(4)-1

For citation: Viuginov N.Y., Fondaratov V.S. Automated Type Contracts Generation in Ruby. *Trudy ISP RAN/Proc. ISP RAS*, 2017, vol. 29, issue 4, pp. 7-20. DOI: 10.15514/ISPRAS-2017-29(4)-1

1. Introduction

Developers suffer from time-consuming investigations when trying to understand why a particular piece of code does not work as expected. The dynamic nature of Ruby allows for great possibilities, which has its drawback: the codebase as a whole becomes entangled and investigations become more difficult compared to statically typed languages like Java or C++ [1]. Another downside of its dynamic features is a drastic reduction in static analysis performance due to inability to resolve some symbols reliably. Consider the dynamic method creation which is often done with *define_method* call. Names and bodies of dynamically created methods may be calculated at runtime [2]. The following code dynamically adds active?, inactive? and pending? methods to the User class:

```
class User
    ACTIVE = 0
    INACTIVE = 1
    PENDING = 2

    attr_accessor :status

    def self.states(*args)
       args.each do |arg|
          define_method "#{arg}?" do
             self.status == User.const_get(arg.upcase)
          end
       end
    end
    states :active, :inactive, :pending
 end
```

One of the possible workarounds to get information about types for such difficult-to-analyze syntactic constructions is using code documentation tools such as RDoc or YARD. @!method annotation defines a method object with a given signature. *@param* and *@return* annotations may help to define the actual types, but they have several drawbacks too:

- the type system used for documenting attributes, parameters and return values is pretty decent, however, it is not clear how to define relations between the types. For example, operator *[]=* for array usually returns the same type as the second arg taking any type so in YARD this will look like *@param value [Object]*, *@return [Object]* which is not really helpful, because all classes in Ruby are inherited from the *Object* and such annotation does not give any additional information about the method.
- from usability perspective, such documentation in some way contradicts the purpose of Ruby to be as short, natural and expressive as possible.

The proposed approach is inspired by the way people tackle this problem manually: one may run or debug the program to inspect the needed info about the code they are investigating. This suggests that collecting direct input and output types of all method dispatches during the program execution with postprocessing and structuring of this data may be considered as a way to automate manual investigations. As a result, it will make up implicit type annotations. As the process is automated, one can retrieve a lot of information about the executed code in the whole project.

Since the quality of the result highly depends on the code coverage of the programs run during the data collection, it is important to be able to merge the result annotations built for the same methods called from different places, projects and even users. These annotations also could be stored in a public database to be shared and reused by different users in order to maximize the coverage of the analyzed code and hence the quality of the generated contracts.

Two main contract generation stages can be distinguished:

- During the first stage, the information about called methods and their input and output types is collected throughout the script execution. It is very important to collect the necessary information as quickly as possible not to keep users waiting for script completion much longer compared to regular execution. To achieve this, we implement a native extension which receives all the necessary information directly from the internal stack of the virtual machine instead of using the standard API provided by the language. *This stage is described in Section 3.*
- During the second stage, the data obtained in the first stage is structured, reduced to a finite-state automaton and prepared for further use in code insight. This storage scheme provides the ability to quickly obtain a regular expression that is easily perceived by a human. *This stage is described in Section 4.*

The generated implicit annotations can be built into the static analysis tools [3] to improve existing and provide additional checks and code completion suggestions. *This stage is described in Section 5.*

2. Related works

In *Static Analysis of Dynamic Languages* [7], static analysis techniques for dynamically and statically typed languages are compared. The author notes that the attributes of dynamically typed languages such as flexibility and expressiveness limit the availability of tool-support for those languages. The paper addresses the main problems of analyzing code written in a language with dynamic typing: particularly, the construction of developer tools is difficult due to the lack of static type systems, therefore, many bugs are not discovered until run-time. The use of static analysis, and in particular whole program dataflow analysis, allow static reasoning about programs written in these languages without changing their nature or imposing unrealistic restrictions on the programmers.

In addition, the article mentions the technique called Use Analysis. “Use Analysis: A heuristic for recovering missing dataflow facts, due to missing library code, by observing how applications objects are used in the application code." An example of such a heuristic is the approach to be described in this article.

For Ruby, as for most dynamically typed languages, there are tools for source code analysis, but they are not capable of statically identifying all errors associated with type mismatch. Here are some of them:

- Rubocop [4] — A Ruby static code analyzer, based on the community-driven Ruby style guide, but it does not allow actual error detection.
- Ruby-lint — A tool for detecting syntax errors, such as undeclared variables, an invalid argument set for calling a method, or unreachable sections of code.
- Diamondback Ruby [5] — an extension to Ruby that aims to bring the benefits of static typing to Ruby. However, at the moment, it is impossible to analyze even the standard Ruby library.

3. Collecting information about method calls

3.1 Calls structure

Method parameters in Ruby have the following structure:

```
def m(a1, a2, ..., aM,                  # mandatory(req)
        b1=(...), ..., bN=(...),       # optional(opt)
        *c,                            # rest
        d1, d2, ..., dL,              # post
        e1:(...), ..., eK:(...),     # keyword
        **f,                         # keyword_rest
        &g)                         # block
```

```
An example of calling this method:
m(11,  12,  21,  22,  1,  2,  3,  '1',  '2',  e1:  1,  e2:  2,  e3:  3)  {...}
# a1   a2   b1   b2   ---c----    d1    d2   e1       e2       f        g
```

TracePoint is an API allowing to hook several Ruby VM events like method calls and returns and get any data through Binding, an object which encapsulates the execution context (variables, methods) and retains this context for the future use.

Consider a simple Ruby method declaration and handlers set for :call and :return events.

```
def foo(a, b = 1)
   b = '1'
end

TracePoint.trace(:call, :return) do |tp|
   binding = tp.binding
   method = tp.defined_class.method(tp.method_id)
   p method.parameters
   puts tp.event, (binding.local_variables.map do |v|
      "#{v}->#{binding.local_variable_get(v).inspect}"
   end.join ', ')
end

foo(2)

The execution output will be:

[[:req, :a], [:opt, :b]]
call
a->2, b->1
[[:req, :a], [:opt, :b]]
return
a->2, b->"1"
```

On each method call, the following information is to be obtained:

- method name
- method receiver class
- arity (names and types of parameters)
- types of arguments and return type, hereinafter "**raw type tuple"**
- name and version of gem (ruby library) in which the method was declared
- location of method declaration

3.2 Unspecified arguments

Code analysis often handles direct method calls, so in order to calculate the return type it is important to distinguish which arguments were directly passed to the method by the user, and which were assigned the default values.

Let the following expression occur during the code analysis: `a, b, c = foo, foo('1'), foo(1),` and the following two contracts be generated: `Int → Int, String → String.` If the method cannot be statically analyzed, then we cannot select a contract to apply to the method call without arguments.

Note that default values are assigned to unspecified optional arguments before the :call event is triggered. Therefore, with the standard API, it is impossible to calculate which arguments were passed to the method, and which were not. This poses a problem because it renders detection of the default value types impossible and, therefore, disables the calculation of the expected return type of calls with any optional parameters unspecified. However, one can build a native extension for the Ruby VM[2] and get this information from an internal stack.

Consider a simple Ruby method with an optional parameter and on appropriate bytecode.

```
def foo(a, b=42, kw1: 1, kw2:, kw3: 3)
     #...
end

foo(1, kw1: '1', kw2: '2')
== disasm: #<ISeq:<compiled>@<compiled>>============
0000 trace            1
0002 putspecialobject 1
0004 putobject        :foo
0006 putiseq          foo
0008   opt_send_without_block   <callinfo!mid:core#define_method,   argc:2,
ARGS_SIMPLE>
0011 pop
0012 trace            1
0014 putself
0015 putobject_OP_INT2FIX_O_1_C_
0016 putstring        "1"
0018 putstring        "2"
0020   opt_send_without_block   <callinfo!mid:foo,   argc:3,   kw:[kw1,kw2],
FCALL|KWARG>
0023 leave
== disasm: #<ISeq:foo@<compiled>>===================
...
```

The instruction number 0020, which calls the method *foo*, has information characterizing the number of passed arguments and the list of passed named arguments. Now we need to find a bytecode instruction for the current method dispatch. It is necessary to find the caller control frame and the last executed instruction in this frame. This instruction will correspond to the call of the method that we are interested in.

The big disadvantage of this approach is that the calculation of the full execution context is a time-consuming operation. But later we will only need information about a small part of it. Namely: types of arguments, types and names of method parameters. Creating a native extension for the Ruby VM, which will receive information about the method name directly from YARV instruction list (Fig. 1), will help us to receive information about argument types directly from the internal stack.

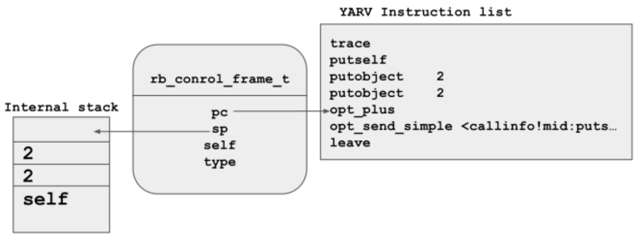


Fig. 1. YARV's internal registers.

4. Tranforming raw call data into contracts

A huge amount of raw data received from the Ruby process must be processed and structured so that it can be easily used and perceived. In our approach, each traced method is associated with a finite-state automaton. This storage structure allows to quickly add raw type tuple obtained from the Ruby process. It can be also easily reduced to a human-readable regular expression.

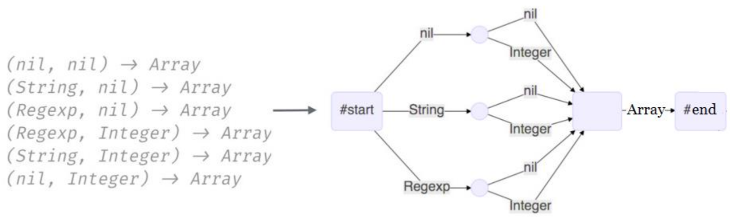


Fig. 2. Example of generating a non-minimized automaton.

In each automaton, there are a single starting vertex, from which the signature begins to be read and a single terminal vertex, in which all edges corresponding to the return types enters. Words obtained by concatenating tuples and corresponding output types are consistently added to the automaton.

```
Data: callArgs, returnType, automaton, parameterList
Result: automaton
$tuple \leftarrow emptyList$
for $param : parameterList$ do
    if $\exists arg : arg \in callArgs \&\& arg.getParam == param$ then
        $tuple << arg$
    else
        $tuple << \epsilon$
    end
end
$node \leftarrow automaton.startVertex$
for $arg : tuple$ do
    $type \leftarrow arg.type$
    if $(node, type) \notin automaton$ then
        $automaton(node, type) \leftarrow newNode$
    end
    $node \leftarrow automaton(node, type)$
end
$automaton(node, returnType) \leftarrow automaton.termVertex$
```

Algorithm 1. Adding a tuple to the automaton

Then, the minimization algorithm [7] is applied to the automaton, but it is slightly modified for the automaton of this type (Alg. 2). Note that all the tuples added to the automaton have the same length, so the resulting automaton has a layered structure based on the distance from the starting vertex. And all the edges emerging from the vertices of the i-th layer go to the vertices of i+1-st layer. Note that, after adding a signature to a minimized automaton, each added vertex can be combined only with the vertex of its level (Fig. 3).

Theorem 1. *Only vertices from the same level can be joined during the minimisation.*

Proof. *Consider two vertices a and b from levels i and j $(i \neq j)$. The vertices a and b join iff their transition functions coincide. All transitions from the vertices of level i lead to vertices of level $i+1$, so transitions from a lead to vertices of level $i+1$, and transitions from vertex b lead to vertices of level $j+1$. It follows from the fact that the vertices adjacent to a and the vertices adjacent to b lie on different levels that the transition functions for the vertices a and b do not coincide.* □

Corollary 1. *Let n be the number of layers of the automaton, then the computational complexity of the minimization algorithm after adding one tuple to the previously minimized automaton can be estimated as: $O(\sum_{i=1}^{n} automaton.levels[i])$ or $O(automaton.size)$, which is better than $O(automaton.size * n)$, as for the automaton in the general case.*

```
**Data**: automaton, nodes
**Result**: automaton
$levels \leftarrow automaton.levels$//splitting automaton for layers
for $node : nodes,\ i{+}{+}$ do
    for $nodeForComparison : levels[i]$ do
        if $node.getTransitions = nodeForComparison.getTransitions$ then
            automaton.joinNodes(node, nodeForComparison);
        end
    end
end
```

Algorithm 2. Automaton minimisation

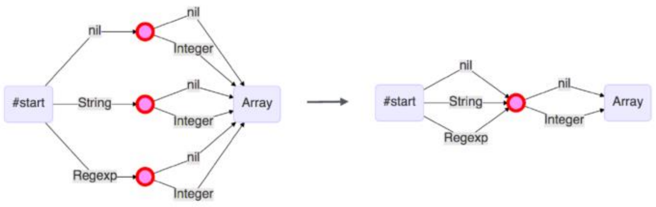


Fig. 3. Joining vertices

Quite often there are situations where types of two or more arguments of the method always coincide or the type of the result coincides with the type of one of the arguments. Consider method *equals* as an example.

```
def equals(a, b)
  raise StandardError if a.class != b.class
  a == b
end
p equals(1, 1)    # (Integer, Integer) —> TrueClass
p equals(1, 2)    # (Integer, Integer) —> FalseClass
p equals(:b, :a)  # (Symbol, Symbol) —> FalseClass
p equals(:a, :a)  # (Symbol, Symbol) —> TrueClass
...
```

While adding the next transition from the vertex to the automaton, let’s compare the symbol of the transition we want to add with all the previous symbols of the current tuple. In case there is at least one match, instead of a regular edge with a type symbol, edge with a bit mask is added. The length of this mask equals to the ordinal number of the current type within the tuple decreased by 1. i-th bit is 1 iff the i-th type in the tuple equals to the type to be added (Fig. 4).

```
**Data**: tuple, type, typeIndex
**Result**: mask
$mask = 0$
**for** $i : [1..typeIndex - 1]$ **do**
    **if** $tuple[i] == type$ **then**
        $mask[i] = 1$
    **end**
**end**
```

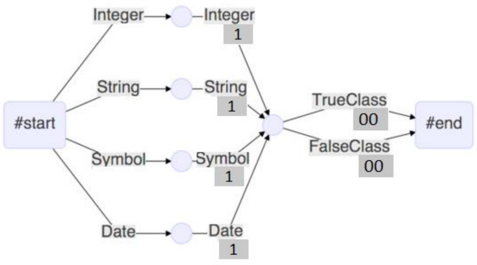


Fig. 4. Automaton with counted bit masks

When reading the signature, each following type is compared to the previous signature types and if a nonzero mask is obtained, one goes through the transition with the mask received.

```
**for** $arg : tuple, i++$ **do**
    $type \leftarrow arg.type$
    $mask \leftarrow calculate\_mask(tuple, type, i)$
    **if** $mask > 0$ **then**
        **if** $(node, mask) \notin automaton$ **then**
            $automaton(node, mask) \leftarrow newNode$
        **end**
        $node \leftarrow automaton(node, mask)$
    **else**
        **if** $(node, type) \notin automaton$ **then**
            $automaton(node, type) \leftarrow newNode$ $node \leftarrow automaton(node, type)$
        **end**
        $node \leftarrow automaton(node, type)$
    **end**
**end**
$automaton(node, returnType) \leftarrow automaton.termVertex$
```

Algorithm 1'. Adding a tuple to the automaton with masks

Theorem 2. *Before the minimization from the vertex cannot be the transition with a type symbol and the transition with an appropriate mask simultaneously.*

Proof. Consider two cases: the transition with the symbol was added before the transition with the mask and vice versa.

1) If an transition with a symbol was added before the transition with the mask, then when it was added, a non-zero mask should have been produced. Then instead of the usual transitions had to be added a transition with a mask.

2) If the transition with a mask was added first, then instead of adding an edge with a symbol, we should just go through the existing transition with mask. □

Corollary 1. *After minimization, the automaton with masks remains deterministic, that is, for every vertex and any type it is impossible to find both conventional and mask edges corresponding to that type simultaneously.*

Automata received from different users need to be merged. The following algorithm is used for this:

```
Data: automaton, additionalAutomaton
Result: automaton
$bfsQueue.push(automaton.getStartNode, additionalAutomaton.getStartNode)$
while $!bfsQueue.empty$ do
    $(oldNode, newNode) = bfsQueue.pop$
    for $transition : newNode.getTransitions$ do
        $node = createNewNode$
        $oldNode.addTransition(transition, node)$
        if $transition \in oldNode.getTransitions$ then
            $nodeToClone = oldNode.goByTransition(transition)$
            $node.getTransitions.add(nodeToClone.getTransitions)$
        end
        $nodes = (node, newNode.goByTransition(transition))$
        if $!used(nodes)$ then
            $used.add(nodes)$
            $bfsQueue.push(nodes)$
        end
    end
end
$automaton.minimize$
```

Algorithm 3. Automatons merge

In Ruby, Duck Typing [8] is quite heavily used. As a consequence, variables of various types that implement a set of methods can be passed as arguments to a method. Hence, many multiple edges corresponding to these classes appear in the automaton. These multiple edges can be replaced by one edge containing information about the interface that all these classes satisfy. Then, to jump on this edge, the next type from the signature must implement this interface. In case this common interface is empty on the edge, it is enough to write the type Object, since it is the parent class for all objects.

5. Using of contracts in static analysis algorithms

The contract is used to calculate the type returned when the method is called with a certain set of arguments. It is worth noting that the types of arguments are not always uniquely defined. Sometimes there is a set of types to which the variable may belong. To calculate the type returned by the method, it is necessary to go successively along the edges of the automaton calculating a set of vertices reachable by reading some sequence of types. The unspecified optional arguments types are imitated with a special non-alphabetic character so that the length of a tuple is lower than the automaton height by 1.

```
Data: argumentTypes, automaton
Result: returnType
$node \leftarrow automaton.startVertex$
$index = 0$
for $type : argumentTypes, index++$ do
    $mask = calculate_mask(argumentTypes$, type, index)
    if $mask \in node.getTransitions$ then
        $node = node.goByTransition(mask)$
        continue
    end
    if $type \in node.getTransitions$ then
        $node = node.goByTransition(type)$
        continue
    end
    return UNKNOWN_SET_OF_ARGUMENTS
end
$returnType << node.getTransitions.types$
```

Algorithm 4. Output type calculation

The generated contracts complement the type selection system because they allow to calculate the types returned from methods which were not successfully analyzed using standard tools. This expands the class of variables for which it is possible to statically compute a type.

The collected information for the methods makes it possible to significantly accelerate the existing control flow analysis because the methods for which a sufficiently representative contract is generated do not require additional analysis. Contracts allow to extend the applicability of some of the features that are supported in most modern IDEs. The functions considered are applicable to method calls for which it was possible to select the class of the object to which they were applied and for this class there is a contract corresponding to the method with that name and configuration of parameters. Functions in which contracts are applied:

- **Go To Declaration/Find Usages.** At the execution time information about method declaration was collected. This information can be used for navigation from method call to declaration and vice versa.
- **Autocompletion.** A list of methods implemented for an object can be supplemented with methods for which the contract was found.
- **, Incorrect method arguments' Inspection**. Information about the method parameters can be used to detect incorrect calls.

6. Conclusion

The paper describes the approach to the generation of implicit type contracts. This approach provides information containing type signatures of methods that cannot be obtained by static analysis using the source code given it is possible to understand in which library the method was declared and to resolve the method receiver. This approach is useful for analyzing programs which heavily utilize dynamic features like dynamic methods creation or when there are complex syntactic constructions in methods implementations. In addition, this approach can be applied to other languages with dynamic typing, such as Python or JavaScript.

Several problems remain unsolved, such as Duck Typing and handling an ambiguous resolve of the argument type in a static analysis.

The problem with duck typing is that, during the execution of the program, it is impossible to save all the methods implemented for the object. Therefore, it is difficult to find the largest common interface for a group of classes.

The problem with arguments with types ambiguous according to the static analysis is that they cannot be read in the automaton.

References

[1]. Brianna M. Ren., J. Toman, T. Stephen Strickland and Jeffrey S. Foster. The ruby type checker. Available: http://www.cs.umd.edu/~jfoster/papers/oops13.pdf

[2]. blog.codeclimate. Gradual type checking for ruby, 2014. [Online]. Available: blog.codeclimate.com/blog/2014/05/06/gradual-type-checking-for-ruby/

[3]. O. Shivers. Control flow analysis in scheme. ACM SIGPLAN 1988 conference on Programming language design and implementation, 1988.

[4]. Bozhidar Batsov. Rubocop, 2017. [Online]. Available: http://batsov.com/rubocop/

[5]. Jeff Foster, Mike Hicks, Mike Furr, David An. Diamond-back ruby guide, 2009.[Online]. Available: http://www.cs.umd.edu/projects/PL/druby/manual/manual.pdf

[6]. Pat Shaughnessy. Ruby Under a Microscope. No Starch Press, 2013.

[7]. Madsen M. Static Analysis of Dynamic Languages. Available: http://pure.au.dk/ws/files/85299449/Thesis.pdf

[8]. Duck Typing [Online]. Available: http://rubylearning.com/satishtalim/duck_typing.html

Автоматизированная генерация типовых контрактов для языка Ruby

[1, 2] Н. Ю. Вьюгинов <viuginov.nickolay@gmail.com>
[2] В. С. Фондаратов <fondarat@gmail.com>
[1] СПбГУ,
199034, Россия, Санкт-Петербург, Университетская наб., 13B
[2] JetBrains,
199034, Россия, Санкт-Петербург, Университетская наб., 7-9-11

Аннотация. Элегантный синтаксис языка Ruby заметно усложняет поиск ошибок в больших кодовых базах. Статический анализ усложняется специфическими возможностями языка, такими как динамическое создание методов и исполнение строковых выражений. Даже в языках с динамической типизацией информация о типах важна, так как она позволяет улучшить типобезопасность и производить более надёжные статические проверки того, определён ли метод для объекта и передан ли метода корректный набор аргументов. Одним из путей решения проблемы является использование YARD нотаций. Они позволяют задокументировать входные и выходный типы методов или даже декларировать методы, добавляемые динамически.

Такие аннотации позволяют улучшить анализ кода и автодополнение. В статье описывается новый подход к генерации типовых аннотаций. Мы отслеживаем непосредственные вызовы метода во время исполнения программы и сохраняем типы аргументов и выходной тип. На основе собранной информации для каждого метода строится неявная типовая аннотация. Каждому автомату сопоставляется конечный автомат, составленный из различных типовых сигнатур метода. К автомату применяется эффективный алгоритм минимизации с целью снизить затраты на хранение и позволяет привести автомат к виду, который может быть легко представлен в виде регулярного выражения. В сгенерированном автомате учитывается только та функциональность метода, которая была покрыта программой, которую исполнил пользователь. Поэтому в подходе предусмотрено объединение автоматов, полученных у разных пользователей с целью увеличения репрезентативности и покрытия функциональности метода.

Ключевые слова: Ruby; динамически типизированные языки; Ruby VM; YARV; сигнатура метода; наследование типов; статический анализ кода

DOI: 10.15514/ISPRAS-2017-29(4)-1

Для цитирования: Вьюгинов Н. Ю., Фондаратов В. С. Автоматизированная генерация типовых контрактов для языка Ruby. Труды ИСП РАН, том 29, вып. 4, 2017 г., стр. 7-20 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-1

Список литературы

[1]. Brianna M. Ren., J. Toman, T. Stephen Strickland and Jeffrey S. Foster. The ruby type checker. Доступно по ссылке: http://www.cs.umd.edu/~jfoster/papers/oops13.pdf
[2]. blog.codeclimate. Gradual type checking for ruby, 2014. [Online]. Доступно по ссылке: blog.codeclimate.com/blog/2014/05/06/gradual-type-checking-for-ruby/
[3]. O. Shivers. Control flow analysis in scheme. ACM SIGPLAN 1988 conference on Programming language design and implementation, 1988.
[4]. Bozhidar Batsov. Rubocop, 2017. [Online]. Доступно по ссылке: http://batsov.com/rubocop/
[5]. Jeff Foster, Mike Hicks, Mike Furr, David An. Diamond-back ruby guide, 2009.[Online]. Доступно по ссылке: http://www.cs.umd.edu/projects/PL/druby/manual/manual.pdf
[6]. Pat Shaughnessy. Ruby Under a Microscope. No Starch Press, 2013.
[7]. Madsen M. Static Analysis of Dynamic Languages. Доступно по ссылке: http://pure.au.dk/ws/files/85299449/Thesis.pdf
[8]. Duck Typing [Online]. Доступно по ссылке: http://rubylearning.com/satishtalim/duck_typing.html

Using Interface Patterns for Compositional Discovery of Distributed System Models[1]

R.A. Nesterov <ranesterov@edu.hse.ru>
I.A. Lomazova <ilomazova@hse.ru>
National Research University Higher School of Economics,
20 Myasnitskaya Ulitsa, Moscow, 101000, Russia

Abstract. Process mining offers various tools for studying process-aware information systems. They mainly involve several participants (or agents) managing and executing operations on the basis of process models. To reveal the actual behavior of agents, we can use process discovery. However, for large-scale processes, it does not yield models, which help understand how agents interact since they are independent and their concurrent implementation can lead to a very sophisticated behavior. To overcome this problem, we propose interface patterns, which allow getting models of multi-agent processes with a clearly identified agent behavior and interaction scheme as well. The correctness of patterns is provided via morphisms. We also conduct a preliminary experiment, results of which are highly competitive compared to the process discovery without interface patterns.

Keywords: Petri nets; interface patterns; synchronization; composition; morphisms; process discovery; multi-agent systems; distributed systems.

DOI: 10.15514/ISPRAS-2017-29(4)-2

For citation: Nesterov R.A., Lomazova I.A. Using Interface Patterns for Compositional Discovery of Distributed System Models. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 21-38. DOI: 10.15514/ISPRAS-2017-29(4)-2

1. Introduction

Process mining is the relatively new direction in studying process-aware information systems. They include information systems managing and executing operational processes, which involve people, applications and information resources through process models [1]. Examples of these systems include workflow management systems, business process management systems, and enterprise

[1]This work is supported by the Basic Research Program at the National Research University Higher School of Economics and Russian Foundation for Basic Research, project No.16-01-00546.

information systems. The underlying interactions among participants (also called agents) of process-aware information systems are intrinsically distributed multiagent systems. An agent acts autonomously, but it can interact with the others via shared resources, restrictions, and other means. Process mining helps to extract a model of this system for further study from a record of its implementation called an event log. However, extracted models are hard for analysis since there might be complex interactions among process participants the number of those can be significant.

In this paper, we propose a compositional approach to address this problem. Given an event log of a distributed system, we can filter it by agents and mine a model of each agent. Then, agent models can be composed to get a complete model of a multi-agent distributed system, which might be simulated. Composing agent models allows us to obtain more structured models compared to models extracted from complete logs since the behavior of an agent can be clearly identified. We compose agent models via interface patterns, which describe how they intercommunicate. This approach was presented at TMPA-2017 [2], the conference proceedings will be available later. The formal proof of the composition correctness is based on using net morphisms [3]. Moreover, interface patterns allow us to inherit deadlock-freeness and proper termination from agents by construction.

We conduct a preliminary experiment on using one interface pattern for mining multi-agent models. The outcomes are evaluated with the help of conformance checking quality dimensions [1, 4] and complexity metrics proposed in [5].

This paper is structured as follows. The next section provides an overview of process discovery and compositional approaches. In Section 3 we introduce basic terms which are used in the paper. Section 4 shows a general description of the compositional approach to process discovery. Section 5 briefly introduces how we compose agent models using interface patterns and net morphisms. In Section 6 we describe the preliminary experiment and analyze results.

2. Related Work

There exist three types of process mining, namely discovery, conformance, and enhancement. Process discovery produces a process model out of an event log – a record of implemented activities. Existing discovery approaches can yield a model in a variety of notations including Petri nets, heuristic nets, process trees, BPMN, and EPC. Petri nets are the most widespread process model representations discovered from event logs. Conformance checking is used to check whether a discovered model corresponds to an input event log and to identify probable deviations. The main idea of enhancement is to improve existing processes using knowledge of actual processes (usually denoted AS-IS) obtained from event logs.

Process discovery offers several methods to be used for constructing models from event logs. One of the first and the most straightforward discovery approach is **α-algorithm,** which identifies ordering relations among activities in logs, but it has severe usage limitations connected with cycles and the overall quality of obtained models [1]. It has several refined versions and improvements, for example [6], but there are other more sophisticated and efficient discovery algorithms. S. Leemans et al. [7] has proposed **inductive miner** allowing to extract process models from logs containing infrequent or incomplete behavior as well as dealing with activity lifecycle when there are separate actions of start and finish for each activity. Apart from that, inductive miner always produces well-structured models in the form of Petri nets. **HeuristicsMiner** is another process discovery algorithm proposed by A. Weijters et al. [8]. It can process event logs with a lot of noise (excessive activities) and also deals with infrequent process behavior. HeuristicsMiner uses intermediate casual matrices and produces heuristics net, which can easily be converted into Petri nets and applied for other notations including EPC, BPMN, and UML. S. van Zelst et al. [9] proposed the approach to process discovery based on **integer linear programming** and **theory of regions**. Their algorithm can produce Petri nets with complex control flow patterns, and its recent improvements guarantee the structural correctness of discovered models. C. Gunther and W. van der Aalst have proposed adaptive **fuzzy mining** approach [10] to deal with unstructured processes extracted from event logs since they can produce different abstractions of processes distinguishing “important” behavior.

Since state-of-the-art process discovery algorithms can deal with complex process behavior, the other problem is to obtain models that are appropriate concerning their structure. A good process model is readable and well-structured, i.e. there is no redundant elements or unnecessary structural complications. There is a so-called continuum of processes ranging from highly structured processes (Lasagna models) to unstructured processes (Spaghetti models) [1]. The problem of obtaining well-structured models is extensively studied in the literature. Researchers offer different techniques to improve model structure [11], and to produce already well-structured process models [12, 13, 14]. In the case of multi-agent and distributed systems using well-structured models should also allow us to identify agent behavior clearly for the model understandability improvement.

We suggest discovering models of agents independently and then composing them together to produce a structured multiagent system model with the clearly visible behavior of each agent. Several compositional approaches for process discovery have been proposed. In [15] A. Kalenkova et al. have shown how to obtain a more readable model from an event log by decomposing extracted transition systems. A special technique to deal with cancellations in process implementation and to produce clear and structured process models which can contain cancellations have been studied in [16]. Also, in [17] authors have proposed a technique for compositional process discovery based on localizing events using region theory to improve overall quality of discovered models.

Correct coordination of system components is an error-prone task. Their interaction can generate complex behavior. The majority of process discovery tools produce Petri nets, and a large amount of literature has investigated the problem of composing Petri nets. They can be composed via straightforward merging of places and transitions [18], but the composition result will not preserve component properties. One of the possible ways to achieve inheritance of component behavioral properties is to use morphisms [19]. Special constructs for composing Petri net based on morphisms were studied in [3, 20, 21]. The key idea of this approach is that distributed system components refine an abstract interface describing the interactions between them. In [22] I. Lomazova has proposed a compositional approach for a flexible re-engineering of business process by using a system of interacting workflow nets. There also exists a several techniques for compositional synthesis of web services [23].

However, in [24] R. Hamadi and B. Benatallah have proposed an algebraic approach to the regular composition of services. These compositional approaches do not let specify the explicit order of inner behavior of two interacting components. This situation is schematically represented in Fig. 1. Having two discovered component models with always executable actions A and B, we want to require that they interact in a way that A is implemented before B. This way of intercommunication is also shown in the form of Petri net.

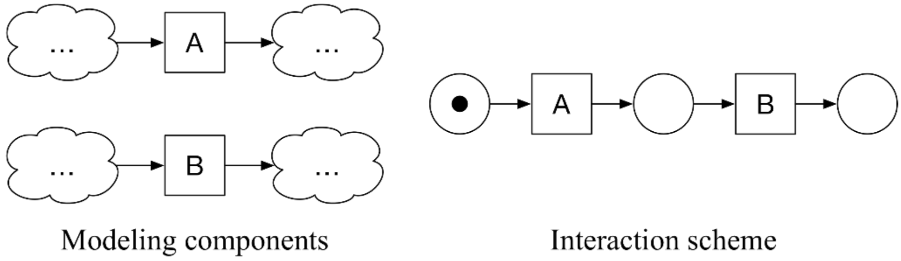


Fig.1. Defining relations on inner actions of components

In [2] we have proposed a solution to this problem and two other patterns for composing two interacting components. The obtained composition inherits properties, such as deadlock-freeness and proper termination, from components.

In this paper, we show how these patterns can be used for discovering a multi-agent system model from an event log in a compositional way. Applying compositional patterns allows us to obtain a more readable model improving time complexity due to the parallelization of process discovery.

We can assess process models obtained from event logs against four standard quality dimensions – fitness, precision, generalization, and simplicity [4]. Fitness identifies how accurately an extracted model can replay a source event log. Precision indicates a fraction of a behavior allowed by the model but not seen in the event log. Generalization tries to measure the extent to which the model will be able to implement the behavior of the process unseen so far in the log. Simplicity focuses on assessing structural complexity alongside with other graph characteristics – a number of elements and a structuredness measure [5].

3. Preliminary Definitions

3.1. Petri Nets

We use Petri nets [18] to represent agent models and an interaction scheme called interface.

Definition 1: A multiset m over a set S is a function $m\colon S \rightarrow \mathbb{N}\cup\{0\}$. Let m and m_0 be two multisets, $m_0 \subseteq m$ iff $\forall s\in S$: $m_0(s)\leq m(s)$. Also, $\forall s\in S$: $(m+m_0)(s)=m(s)+m_0(s)$ and $(m–m_0)(s)=max(0, m(s), m_0(s))$.

Then, an ordinary set is a multiset in which distinct elements occur only once.

Definition 2: A Petri net is a bipartite graph $N=(P, T, F, m_0, L)$, where:

1. $P=\{p_1, p_2, ..., p_n\}$ – a finite non-empty set of places.
2. $T=\{t_1, t_2, ..., t_m\}$ – a finite non-empty set of transitions, $P\cap T=\emptyset$.
3. $F \subseteq (P\times T)\cup(T\times P)$ – a flow relation.
4. m_0: $P \rightarrow \mathbb{N}\cup\{0\}$ – a multiset over P, initial marking.
5. L: $T \rightarrow \mathcal{A}\cup\{\tau\}$ – a labeling for transitions, where τ is a name for silent transitions.

Pictorially, places are shown as circles, and transitions are shown as boxes (silent transitions are depicted by black boxes). A flow relation is depicted by directed arcs (see Fig. 2).

Let $X=P\cup T$. We call a set ${}^\bullet x=\{y\in X \mid (y,x)\in F\}$ a **preset** of x and a set $x^\bullet=\{y\in X \mid (x,y)\in F\}$ – a **postset** of x. *Also* ${}^\bullet x^\bullet={}^\bullet x\cup x^\bullet$ is a neighborhood of x.

The behavior of Petri nets is defined by the firing rule, which specifies when an action can occur, and how it modifies the overall state of the system.

A marking m: $P \rightarrow \mathbb{N}\cup\{0\}$ enables a transition t, denoted $m[t\rangle$, if ${}^\bullet t\subseteq m$. The t firing at m leads to m', denoted $m[t\rangle m'$, where $m'=m-{}^\bullet t+t^\bullet$. When $\forall t\in T$ *and* $\forall w\in T^*$, $m[tw\rangle m'=m[t\rangle m''[w\rangle m$, w is then called a firing sequence. We denote a set of all firing sequences of a net N as $FS(N)$.

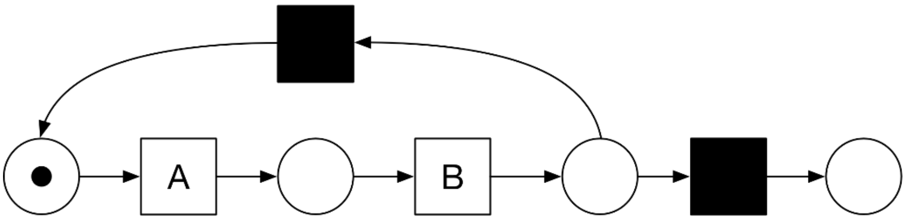


Fig.2. A Petri net with silent transitions

We call a marking m reachable from m_0 if $\exists w\in FS(N)$: $m_0[t\rangle m$. A set of all markings reachable from m_0 is denoted by $[m_0\rangle$. So, $[m\rangle$ is a set of all markings reachable from m. A net N is safe if $\forall p\in P$, $\forall m_0\in[m_0\rangle$: $m(p)\leq 1$.

A marking m_f is called final if $\forall p \in m_f : p^{\bullet} = \emptyset$. A net N is deadlock-free if $\forall t \in T$ $\exists m \in [m_0\rangle$: $m[t\rangle$ and $m \neq m_f$. A net N terminates properly if a final marking is reachable from all reachable states $\forall m \in [m_0\rangle$: $m_f \in [m\rangle$.

3.1. Event Logs

Process discovery techniques allow generating process models from event logs containing information on executed actions. In a simple case, event logs may contain actions names and a corresponding implementation order. We can augment this record with a timestamp (when an action occurs) and executor (what agent implements it).

Definition 3: Let $\mathcal{N}$ be a set of action names and $\mathcal{E}$ be a set of agent names. An activity is a triple (n, e, t), where $n \in \mathcal{N}$, $e \in \mathcal{E}$, and t corresponds to a timestamp. The set of all activities is denoted by Act. A trace $\sigma \in Act^+$ is a sequence of activities. An event log L is a multiset over Act^+, $L \in m(Act^+)$.

Different traces can be combined to form a case corresponding to a process implementation scenario. XES is a standard representation format adopted by IEEE [25] for logging events and processing them via process mining tools.

Table 1. A fragment of an event log

Trace ID	Action ID	Timestamp	Executor
Trace 1			
	t_1	2017-03-01T17:23:40	Agent 1
	e_2	2017-03-01T19:12:05	Agent 2
	…	…	…
Trace 2			
	e_1	2017-03-02T21:13:47	Agent 2
	t_1	2017-03-04T21:14:40	Agent 1
	…	…	…

4. Compositional Process Discovery

4.1. General Outline

To support the compositional discovery of models from event logs generated by multi-agent systems, we assume a record of each action has a corresponding label of an agent implementing it. The procedure of the compositional synthesis includes several steps to be implemented:

1. Capturing a complete event log L from multi-agent system operation.
2. Filtering the event log L by agent labels and producing a set of event logs L_e ($|L_e| = |\mathcal{E}|$), each trace consists of actions implemented by e only.

3. Discovering a model for each agent separately from the set of event logs L_e;
4. Defining interface pattern which describes how agents intercommunicate;
5. Composing agent models and producing a multi-agent system model.

The step of defining interface pattern for agent interaction is implemented manually so far. We rely on an expert view on how agents should intercommunicate.

4.2. Software Overview

A wide range of process discovery tools is implemented within the context of the open-source project **ProM** [26] continuously improving nowadays. However, there also exist many commercial tools using process mining approach to analyze and improve business process. They include **Disco** [27], **QPR ProcessAnalyzer** [28], **myInvenio** [29] to name but a few. Contrary to ProM, they provide more business-related solutions for process performance analysis and further improvement.

To process event logs we use the advanced ProM plugin **GENA** [30] which allows to generate event logs with timestamps and originator labels as well as to augment logs with artificial events representing noise.

5. Composing Petri Nets via Interface Patterns

This section provides a brief introduction to our approach to Petri net composition using interfaces and net morphisms.

5.1. Composing Petri Nets via morphisms

The notion of ω-morphism on Petri nets was first introduced in [3] for elementary net systems and can be applied for safe nets.

Definition 4: Let $N_i = (P_i, T_i, F_i, m_0^i, L_i)$ be two safe Petri nets for $i=1,2$. The ω-morphism is a total surjective map φ: $N_1 \rightarrow N_2$ such that:

1. $\varphi(P_1)=P_2$.
2. $\forall t_1 \in T_1: \varphi(t_1) \in T_2 \Rightarrow \varphi({}^{\bullet}t_1) = {}^{\bullet}\varphi(t_1) \land \varphi(t_1^{\bullet}) = \varphi(t_1)^{\bullet}$.
3. $\forall t_1 \in T_1: \varphi(t_1) \in P_2 \Rightarrow \varphi({}^{\bullet}t_1^{\bullet}) = \{\varphi(t_1)\}$.
4. $\forall p_1 \in P_1: m_0^1(p_1) > 0 \Rightarrow m_0^2(\varphi(p_1)) = m_0^1(p_1)$.

Figure 3 helps to explain requirements 2 and 3 of the definition. i.e. how transitions of N_1 can be mapped onto places and transitions of N_2.

To use morphisms for Petri net composition, we need to define morphisms from agent nets towards an interface net, which describes how they intercommunicate. Then we merge transitions having common labels and images. Figure 4 shows how two Petri nets are composed via ω-morphisms represented as dotted arrows.

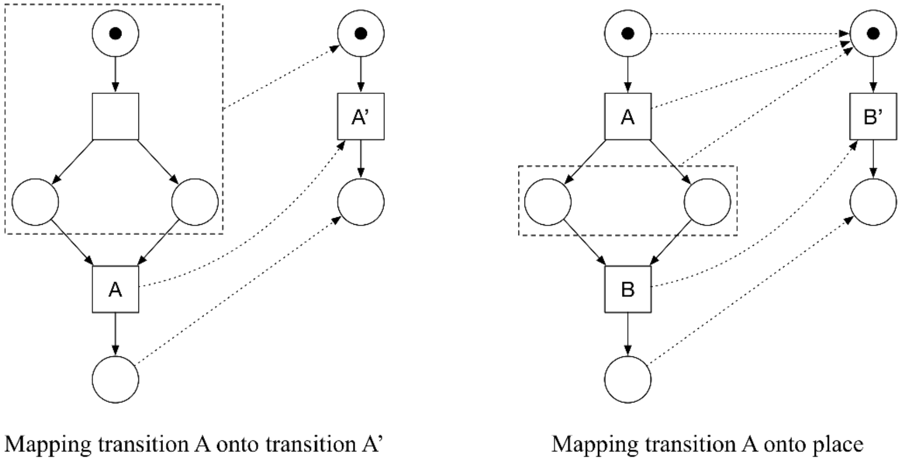

A'
A
B'
A
B
Mapping transition A onto transition A'
Mapping transition A onto place

Fig.3. Transition map options for ω-morphism

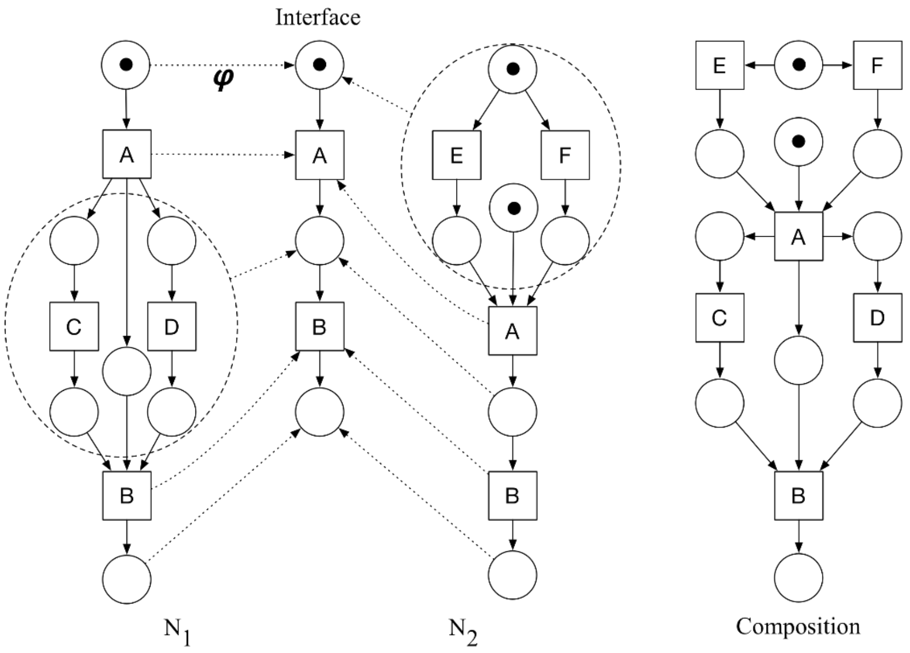


Fig.4. Composing two Petri nets via ω-morphism

As it was proved in [19], the use of morphisms allows us to preserve properties of interacting components in a composed process net. A composition obtained via ω-morphisms is deadlock-free and properly terminates iff source component nets and interface net are deadlock-free and terminate properly as well.

5.2. Compositional Interface Patterns

To facilitate Petri net composition, we use compositional patterns for typical interface we have proposed in [2]. One of such patterns called the simple causality is schematically shown in Fig. 1, and Fig. 5 provides its instantiation. A pattern includes component and interface net which might be merged according to the morphism composition rules if there is a need to produce a model for comprehensive simulation.

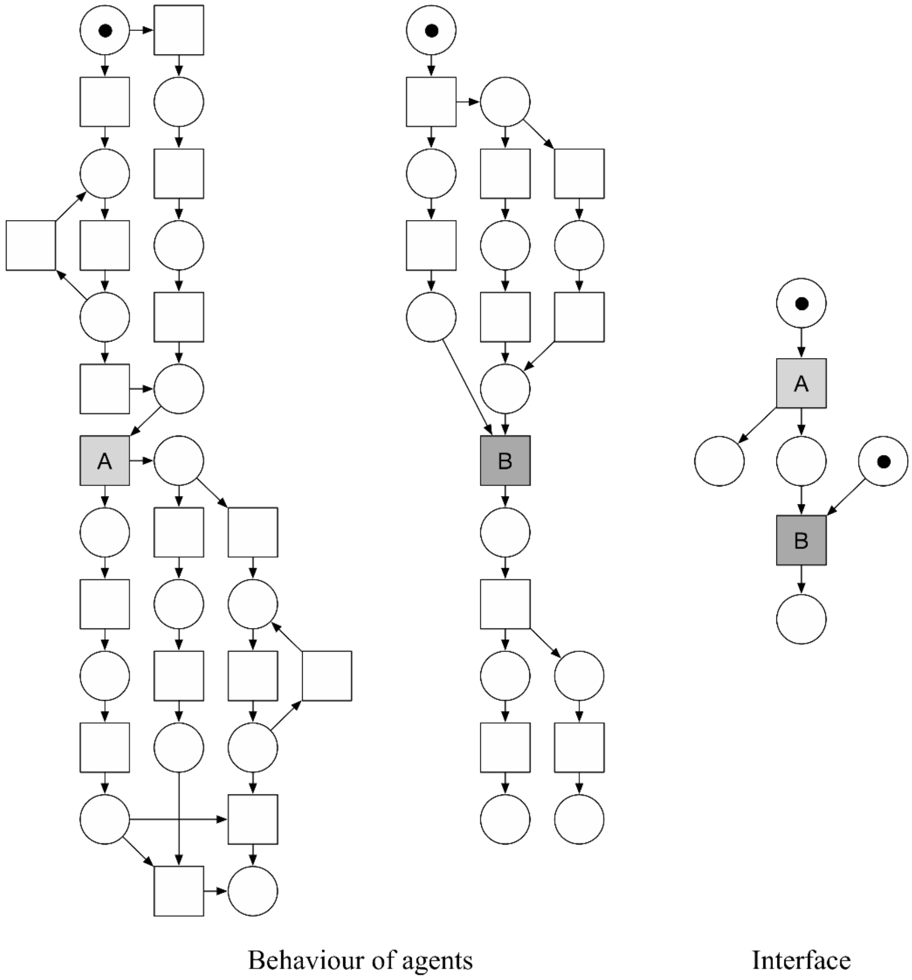


Fig.5. Instantiating simple causality pattern

It also has to be mentioned that to preserve concurrency in the implementation of interacting agents we expand interface nets with additional places and transitions keeping them weakly bisimilar with original interfaces. Consequently, extended interfaces allow us to obtain composition results with the clearly identified behavior of each component.

Figure 5(b) shows how we have expanded interface net for this pattern. We use expanded interfaces only for our inner purposes. The end user does not need to know the underlying theoretical aspects of our approach.

6. Some Experimental Evaluation

In this section, we describe a preliminary experiment on using the simple causality pattern for compositional process discovery. To test our approach we use artificial event logs obtained from the instantiated simple causality pattern. Then we also assess quality metrics of discovered models and provide a balanced consideration.

6.1. Processing Event Logs

Using GENA and the composition result obtained from the instantiated simple causality pattern (see Fig. 5) we have generated the event log with 3000 traces. Then we have filtered the initial log by executors using ProM. The obtained event logs have the characteristics presented in Table 2. Generation results for Agent A show bigger values due to cycles.

Table 2. Characteristics of event logs

	Log *L*	**Log L_A**	**Log L_B**
Number of traces	3000	3000	3000
Number of events	58466	34466	24000
Events per trace (min)	17	9	8
Events per trace (max)	43	35	8
Events per trace (mean)	19	11	8

6.2. Discovering a System Model from Log *L*

Figure 6 shows the fragment of the Petri net discovered from the event log L using Inductive Miner and ProM. The behavior of agents is distinguished by colors.

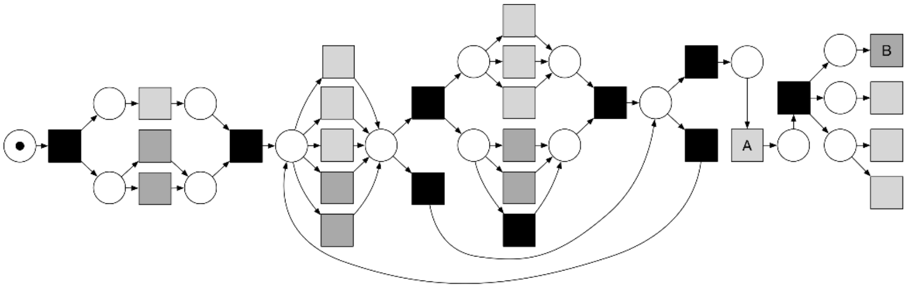


Fig.6. The fragment of the system model discovered from L

This discovered model is quite well-structured (constructed out of clear blocks) but it does not allow to identify the behavior of different agents. That is why, it is hard to yield the complete picture of agent intercommunication scheme.

6.3. Discovering and Composing Models from Logs L_A and L_B

Figure 7 shows the fragment of the composed Petri nets we have discovered from the agent event logs L_A and L_B also using Inductive Miner and ProM. It has to be mentioned that Petri nets discovered by Inductive Miner are always safe. Hence we can apply the approach based on morphisms to compose separately discovered models of agent behavior.

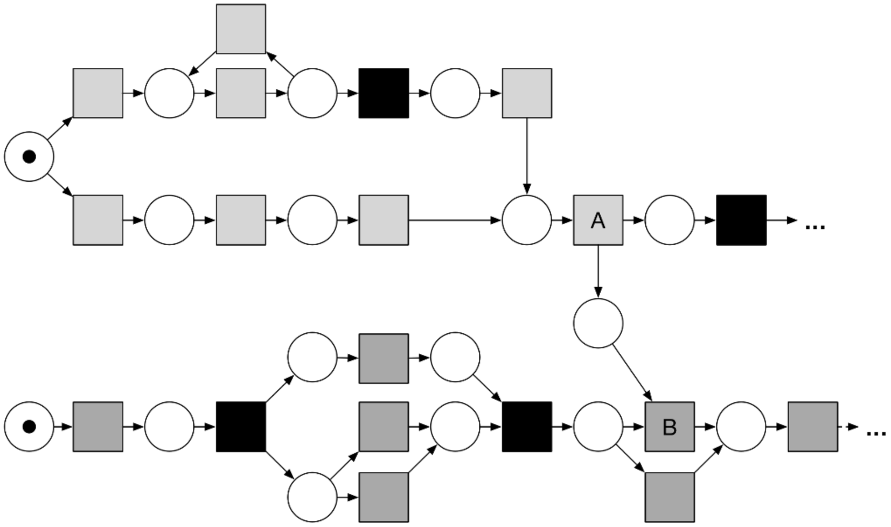


Fig.7. The fragment of the composed system model discovered from L_A and L_B

The merged model allows us to identify the behavior of agents clearly and how they intercommunicate. Using morphisms guarantees inheritance of properties such as deadlock-freeness and proper termination of agents by the entire net.

6.4. Analysis of the Experiment Results

ProM implementation of Inductive miner offers three configuration options:

1. event logs with **infrequent** behavior;
2. event logs with **incomplete** behavior;
3. event logs with **lifecycle** events (start/finish of events);
4. exhaustive k-successor algorithm.

We do not work with incomplete logs or with lifecycle logs for now. So, in our experiment we have discovered models of system and agents shown in previous subsections in accordance with options 1 and 4 and compared them using structural process discovery metrics.

Table 3 provides the comparison of structural characteristics for the directly discovered and composed system models. We have compared obtained models with respect to the number of Petri net elements and structure metric which assess the overall complexity of a model by breaking it into trivial constructs and assigning weights to each reducing step. Models discovered with infrequent configuration are denoted as INFR, models discovered with exhaustive configuration are denoted as EXHS.

The experiment results show the increase in transition numbers because of adding silent transitions. Compositional patterns obviously decrease a number of arcs, compared to direct discovery, as long as we simplify agent intercommunication. Composed models also preserve complex control flows as shown by structuredness measure. Separately discovered agent models and their composition exhibit more precise cycle discovery.

Table 3. Structural analysis of system models

		Direct		**Composed**	
	Source	**INFR**	**EXHS**	**INFR**	**EXHS**
Places	28	30	47	35	39
Transitions	27	44	46	40	41
Arcs	68	100	114	89	93
Structuredness	9360	240	496	872	1208

We have also conducted conformance checking for directly discovered and composed models. As it was mentioned above, there are four standard quality dimensions, namely fitness, precision, simplicity, and generalization. Simplicity is analyzed above via structural analysis. We do not estimate generalization since there are no complex cyclic or concurrent constructs to instantiate the simple causality pattern. Table 4 shows values obtained for fitness and precision of discovered and composed system models.

Table 4. Quality analysis of system models

		Direct		**Composed**	
	Source	**INFR**	**EXHS**	**INFR**	**EXHS**
Fitness	1,0000	1,0000	0,9684	1,0000	1,0000
Precision	0,6992	0,3631	0,5508	0,5629	0,6232

Both discovered and composed system models preserve the appropriate level of fitness, the composition does not block its preservation. What is more important, using compositional patterns produces models with precision nearer to that of the source model compared to direct discovery results. Composed models approximately 30% more precise than discovered ones.

To sum up, we used the simple causality pattern to produce the model of the multi-agent system. Assessment results showed that the composed models are highly competitive with the models directly discovered from complete event logs in the context of their relative structural complexity evaluations and conformance checking results.

7. Conclusion and Future Work

In this paper, we have proposed the solution to the problem of discovering structured models for the processes with several participants (agents). The key idea is to automatically obtain the correct and complete process models from the separate source models of its components. The interaction between agents is defined by experts.

To prove the correctness of the composition we adopt the approach based on Petri net morphisms. We refer to the compositional patterns proposed for the correct synthesis of models for multi-agent processes. In the context of this work, we conducted the preliminary experiment on using the simple causality pattern for constructing the complete model from discovered agent models. The analysis of experimental results (conformance and complexity) showed that composed models are highly competitive compared to the models obtained directly. Moreover, our compositional approach to process discovery allows producing models with the clearly identified behavior of interacting agents.

We aim to continue developing of compositional patterns for typical interfaces and providing experimental process discovery implementations for them using also real-live event logs. Also, we will proceed with complex synchronization patterns with relations on action sets and their correct combinations.

References

[1]. van der Aalst W.M.P. Process Mining: Discovery, Conformance and Enhancement of Business Processes, 1st ed. Springer Publishing Company, Incorporated, 2011. DOI: 10.1007/978-3-642-19345-3.

[2]. Nesterov R.A., Lomazova I.A. Compositional process model synthesis based on interface patterns. Communications in Computer and Information Science, 2017.

[3]. Bernardinello L., Mangioni E., Pomello L. Local state refinement and composition of elementary net systems: An approach based on morphisms. Transactions on Petri Nets and Other Models of Concurrency, 2013, vol. 8, pp. 48–70. DOI: 10.1007/978-3-642-40465-8_3.

[4]. Buijs J.C.A.M., Dongen B., van der Aalst W.M.P. On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. Lecture Notes in Computer Science, 2012, vol. 7565, pp. 305–322. DOI: 10.1007/978-3-642-33606-5_19.

[5]. Lassen K.B., van der Aalst W.M.P. Complexity metrics for workflow nets. Information and Software Technology, 2009, vol. 51, issue 3, pp. 610–626. DOI: 10.1016/j.infsof.2008.08.005.

[6]. Wen L., van der Aalst W.M.P., Wang J., Sun J. Mining process models with non-free-choice constructs. Data Mining and Knowledge Discovery, 2007, vol. 15, issue 2, pp. 145–180. DOI: 10.1007/s10618-007-0065-y.

[7]. Leemans S.J.J., Fahland D., van der Aalst W.M.P. Discovering block-structured process models from event logs containing infrequent behavior. Lecture Notes in Business Information Processing, 2013, vol. 171, pp. 66–78. DOI: 10.1007/978-3-319-06257-0_6.

[8]. Weijters A.J.M.M., van der Aalst W.M.P., de Medeiros A.K.A. Process Mining with the HeuristicsMiner Algorithm. BETA Working Paper Series, 2006, vol. 166, Einhoven University of Technology.

[9]. van Zelst S.J., van Dongen B.F., van der Aalst W.M.P. ILP-based process discovery using hybrid regions. CEUR Workshop Proceedings, 2015, vol. 1731, pp. 47–61.

[10]. Gunther C.W., van der Aalst W.M.P. Fuzzy mining – adaptive process simplification based on multi-perspective metrics. Lecture Notes in Computer Science, 2007, vol 4714, pp. 328–343. DOI: 10.1007/978-3-540-75183-0_24.

[11]. van der Aalst W.M.P., Gunther C.W. Finding structure in unstructured processes: The case for process mining. ACSD '07 Proceedings of the Seventh International Conference on Application of Concurrency to System Design, 2007, pp. 3–12. DOI: 10.1109/ACSD.2007.50

[12]. Buijs J.C.A.M. Flexible Evolutionary Algorithms for Mining Structured Process Models. Ph.D. dissertation, Eindhoven University of Technology, 2014.

[13]. Smedt J.D., Weerdt J.D., Vanthienen J. Multi-paradigm process mining: Retrieving better models by combining rules and sequence. Lecture Notes in Computer Science, 2014, vol. 8841, pp. 446–453. DOI: 10.1007/978-3-662-45563-0_26.

[14]. de San Pedro J., Cortadella J. Mining Structured Petri Nets For The Visualization Of Process Behavior. SAC '16 Proceedings of the 31st Annual ACM Symposium on Applied Computing, 2016, pp. 839–846. DOI: 10.1145/2851613.2851645.

[15]. Kalenkova A.A., Lomazova I.A., van der Aalst W.M.P. Process Model Discovery: A Method Based on Transition System Decomposition. Lecture Notes in Computer Science, 2014, vol. 8489, pp. 71–90. DOI: 10.1007/978-3-319-07734-5_5

[16]. Kalenkova A.A., Lomazova I.A. Discovery of Cancellation Regions within Process Mining Techniques. Fundamenta Informaticae, 2014, vol. 133, issue 2-3, pp. 197–209. DOI: 10.3233/FI-2014-1071.

[17]. van der Aalst, Kalenkova A., Rubin V., Verbeek E. Process Discovery Using Localized Events. Lecture Notes in Computer Science, 2015, vol. 9115, pp. 287–307. DOI: 10.1007/978-3-319-19488-2_15.

[18]. Reisig W. Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies. Springer, 2013, 145 p. DOI: 10.1007/978-3-642-33278-4.

[19]. Winskel G. Petri nets, morphisms and compositionality. Lecture Notes in Computer Science, 1985, vol. 222, pp. 453–477. DOI: 10.1007/BFb0016226.

[20]. Bernardinello L., Monticelli E., Pomello L. On Preserving Structural and Behavioural Properties by Composing Net Systems on Interfaces. Fundamenta Informaticae, 2007, vol. 80, issue 1-3, pp. 31–47.

[21]. Bernardinello L., Pomello L., Scaccabarozzi S. Morphisms on Marked Graphs. CEUR Workshop Proceedings, 2014, vol. 1160, pp. 113–127.

[22]. Lomazova I.A. Interacting Workflow Nets For Workflow Process Reengineering. Fundamenta Informaticae, 2010, vol. 101, issue 1-2, pp. 59–70. DOI: 10.3233/FI-2010-275.

[23]. Cardinale Y., Haddad J.E., Manouvrier M., Rukoz M. Web Service Composition Based On Petri Nets: Review and Contribution. Lecture Notes in Computer Science, vol. 8194, 2012, pp. 83–122. DOI: 10.1007/978-3-642-45263-5_5.

[24]. Hamadi R., Benatallah B. A Petri Net-Based Model For Web Service Composition. ADC‘03 Proceedings of the 14th Australasian database conference, 2003, pp. 191–200.

[25]. XES (eXtensible Event Stream). Available at: http://www.processmining.org/logs/xes, accessed 10.03.2017.

[26]. van Dongen B.F., de Medeiros A.K.A., Verbeek H.M.W., Weijters A.J.M.M., van der Aalst W.M.P. The ProM Framework: A New Era in Process Mining Tool Support. Lecture Notes in Computer Science, 2005, vol. 3536, pp. 444–454. DOI: 10.1007/11494744_25.

[27]. Gunther C.W., Rozinat A. Disco: Discover your processes. CEUR Workshop Proceedings, 2012, vol. 940, pp. 40–44.

[28]. Ailenei I.M. Process Mining Tools: A Comparative Analysis. Master’s thesis, Eindhoven University of Technology, 2011.

[29]. BPM Tool – myInvenio. Available: https://www.my-invenio.com, accessed 10.03.2017.

[30]. Shugurov I.S., Mitsyuk A.A. Generation of a Set of Event Logs with Noise. Proceedings of the 8th Spring/Summer Young Researchers Colloquium on Software Engineering (SYRCoSE 2014), 2014, pp. 88–95.DOI: 10.15514/SYRCOSE-2014-8-13.

Автоматизированный композициональный синтез моделей распределенных систем с помощью паттернов интерфейсов

Р.А. Нестеров <ranesterov@edu.hse.ru>
И.А. Ломазова <ilomazova@hse.ru>
Научно-учебная лаборатория процессно-ориентированных информационных систем (ПОИС),
Национальный исследовательский университет «Высшая школа экономики»
101000, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. Средства и методы process mining позволяют исследовать различные аспекты процессно-ориентированных информационных систем. Как правило, в рамках таких систем несколько исполнителей (агентов) взаимодействуют друг с другом. Поведение агентов, а также механизмы их взаимодействия описываются с помощью моделей процессов. Для моделирования процессов мы применяем обыкновенные сети Петри. Алгоритмы process discovery позволяют восстановить модели реального поведения агентов из журнала событий системы. Однако в случае масштабных систем анализ взаимодействия как поведения отдельных агентов, так и всей системы в целом затруднителен, так как получаемые модели крупномасштабных систем в большинстве случаев крайне громоздкие и плохо читаемые. Для решения этой проблемы мы предлагаем использовать так называемые паттерны интерфейсов, которые описывают, как агенты взаимодействуют друг с другом. С их помощью полная модель мультиагентной системы может быть получена путем композиции отдельных моделей агентов. Кроме того, модели мультиагентных систем, построенные с применением паттернов интерфейсов, позволяет легко идентифицировать поведение каждого отдельного агента. В целях обеспечения корректности применения паттернов интерфейсов мы применяем специальные конструкции на сетях Петри – морфизмы. Результаты эксперимента по применению паттерна для композиционального синтеза модели мультиагентной системы, представленные в работе, показали прирост основных метрик качества по сравнению с моделями, получаемыми с помощью стандартного подхода process discovery.

Ключевые слова: сети Петри; паттерны интерфейсов; синхронизация; композиция; морфизмы; извлечение процессов; мультиагентные системы; распределенные системы.

DOI: 10.15514/ISPRAS-2017-29(4)-2

Для цитирования: Нестеров Р.А., Ломазова И.А. Автоматизированный композициональный синтез моделей распределенных систем с помощью паттернов интерфейсов. *Труды ИСП РАН,* том 29, вып. 4, 2017 г., стр. 21-38 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-2

Список литературы

[1]. van der Aalst W.M.P. Process Mining: Discovery, Conformance and Enhancement of Business Processes, 1st ed. Springer Publishing Company, Incorporated, 2011. DOI: 10.1007/978-3-642-19345-3.

[2]. Nesterov R.A., Lomazova I.A. Compositional process model synthesis based on interface patterns. Communications in Computer and Information Science, 2017.

[3]. Bernardinello L., Mangioni E., Pomello L. Local state refinement and composition of elementary net systems: An approach based on morphisms. Transactions on Petri Nets and Other Models of Concurrency, 2013, vol. 8, pp. 48–70. DOI: 10.1007/978-3-642-40465-8_3.

[4]. Buijs J.C.A.M., Dongen B., van der Aalst W.M.P. On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. Lecture Notes in Computer Science, 2012, vol. 7565, pp. 305–322. DOI: 10.1007/978-3-642-33606-5_19.

[5]. Lassen K.B., van der Aalst W.M.P. Complexity metrics for workflow nets. Information and Software Technology, 2009, vol. 51, issue 3, pp. 610–626. DOI: 10.1016/j.infsof.2008.08.005.

[6]. Wen L., van der Aalst W.M.P., Wang J., Sun J. Mining process models with non-free-choice constructs. Data Mining and Knowledge Discovery, 2007, vol. 15, issue 2, pp. 145–180. DOI: 10.1007/s10618-007-0065-y.

[7]. Leemans S.J.J., Fahland D., van der Aalst W.M.P. Discovering block-structured process models from event logs containing infrequent behavior. Lecture Notes in Business Information Processing, 2013, vol. 171, pp. 66–78. DOI: 10.1007/978-3-319-06257-0_6.

[8]. Weijters A.J.M.M., van der Aalst W.M.P., de Medeiros A.K.A. Process Mining with the HeuristicsMiner Algorithm. BETA Working Paper Series, 2006, vol. 166, Einhoven University of Technology.

[9]. van Zelst S.J., van Dongen B.F., van der Aalst W.M.P. ILP-based process discovery using hybrid regions. CEUR Workshop Proceedings, 2015, vol. 1731, pp. 47–61.

[10]. Gunther C.W., van der Aalst W.M.P. Fuzzy mining – adaptive process simplification based on multi-perspective metrics. Lecture Notes in Computer Science, 2007, vol 4714, pp. 328–343. DOI: 10.1007/978-3-540-75183-0_24.

[11]. van der Aalst W.M.P., Gunther C.W. Finding structure in unstructured processes: The case for process mining. ACSD '07 Proceedings of the Seventh International Conference on Application of Concurrency to System Design, 2007, pp. 3–12. DOI: 10.1109/ACSD.2007.50

[12]. Buijs J.C.A.M. Flexible Evolutionary Algorithms for Mining Structured Process Models. Ph.D. dissertation, Eindhoven University of Technology, 2014.

[13]. Smedt J.D., Weerdt J.D., Vanthienen J. Multi-paradigm process mining: Retrieving better models by combining rules and sequence. Lecture Notes in Computer Science, 2014, vol. 8841, pp. 446–453. DOI: 10.1007/978-3-662-45563-0_26.

[14]. de San Pedro J., Cortadella J. Mining Structured Petri Nets for the Visualization of Process Behavior. SAC '16 Proceedings of the 31st Annual ACM Symposium on Applied Computing, 2016, pp. 839–846. DOI: 10.1145/2851613.2851645.

[15]. Kalenkova A.A., Lomazova I.A., van der Aalst W.M.P. Process Model Discovery: A Method Based on Transition System Decomposition. Lecture Notes in Computer Science, 2014, vol. 8489, pp. 71–90. DOI: 10.1007/978-3-319-07734-5_5.

[16]. Kalenkova A.A., Lomazova I.A. Discovery of Cancellation Regions within Process Mining Techniques. Fundamenta Informaticae, 2014, vol. 133, issue 2-3, pp. 197–209. DOI: 10.3233/FI-2014-1071.

[17]. van der Aalst, Kalenkova A., Rubin V., Verbeek E. Process Discovery Using Localized Events. Lecture Notes in Computer Science, 2015, vol. 9115, pp. 287–307. DOI: 10.1007/978-3-319-19488-2_15.

[18]. Reisig W. Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies. Springer, 2013, 145 p. DOI: 10.1007/978-3-642-33278-4.

[19]. Winskel G. Petri nets, morphisms and compositionality. Lecture Notes in Computer Science, 1985, vol. 222, pp. 453–477. DOI: 10.1007/BFb0016226.

[20]. Bernardinello L., Monticelli E., Pomello L. On Preserving Structural and Behavioural Properties by Composing Net Systems on Interfaces. Fundamenta Informaticae, 2007, vol. 80, issue 1-3, pp. 31–47.

[21]. Bernardinello L., Pomello L., Scaccabarozzi S. Morphisms on Marked Graphs. CEUR Workshop Proceedings, 2014, vol. 1160, pp. 113–127.

[22]. Lomazova I.A. Interacting Workflow Nets For Workflow Process Reengineering. Fundamenta Informaticae, 2010, vol. 101, issue 1-2, pp. 59–70. DOI: 10.3233/FI-2010-275.

[23]. Cardinale Y., Haddad J.E., Manouvrier M., Rukoz M. Web Service Composition Based On Petri Nets: Review and Contribution. Lecture Notes in Computer Science, vol. 8194, 2012, pp. 83–122. DOI: 10.1007/978-3-642-45263-5_5.

[24]. Hamadi R., Benatallah B. A Petri Net-Based Model For Web Service Composition. ADC‘03 Proceedings of the 14th Australasian database conference, 2003, pp. 191–200.

[25]. XES (eXtensible Event Stream). Available at: http://www.processmining.org/logs/xes, accessed 10.03.2017.

[26]. van Dongen B.F., de Medeiros A.K.A., Verbeek H.M.W., Weijters A.J.M.M., van der Aalst W.M.P. The ProM Framework: A New Era in Process Mining Tool Support. Lecture Notes in Computer Science, 2005, vol. 3536, pp. 444–454. DOI: 10.1007/11494744_25.

[27]. Gunther C.W., Rozinat A. Disco: Discover your processes. CEUR Workshop Proceedings, 2012, vol. 940, pp. 40–44.

[28]. Ailenei I.M. Process Mining Tools: A Comparative Analysis. Master’s thesis, Eindhoven University of Technology, 2011.

[29]. BPM Tool – myInvenio. Available: https://www.my-invenio.com, accessed 10.03.2017.

[30]. Shugurov I.S., Mitsyuk A.A. Generation of a Set of Event Logs with Noise. Proceedings of the 8th Spring/Summer Young Researchers Colloquium on Software Engineering (SYRCoSE 2014), 2014, pp. 88–95. DOI: 10.15514/SYRCOSE-2014-8-13.

A contract-based method to specify stimulus-response requirements

[1]A. Naumchev <a.naumchev@innopolis.ru>
[1]M. Mazzara <m.mazzara@innopolis.ru>
[1, 2, 3]B. Meyer <Bertrand.Meyer@inf.ethz.ch>
[3]J.-M. Bruel <bruel@irit.fr>
[3]F. Galinier <galinier@irit.fr>
[3]S. Ebersold <ebersold@irit.fr>
[1]Innopolis University,
1 Universitetskaya st., Innopolis, 420500, Russian Federation.
[2]Politecnico di Milano,
Piazza Leonardo da Vinci, 32, 20133 Milano MI, Italy.
[3]Paul Sabatier University,
118 Route de Narbonne, 31062 Toulouse, France.

Abstract. The verification of many practical systems – in particular, embedded systems – involves processes executing over time, for which it is common to use models based on temporal logic, in either its linear (LTL) or branching (CTL). Some of today's most advanced automatic program verifiers, however, rely on non-temporal theories, particularly Hoare-style logic. Can we still take advantage of this sophisticated verification technology for more challenging systems? As a step towards a positive answer, we have defined a translation scheme from temporal specifications to contract-equipped object-oriented programs, expressed in Eiffel and hence open for processing by the AutoProof program prover. We have applied this scheme to a published CTL model of a widely used realistic example, the "landing gear" system which has been the subject of numerous competing specifications. An attempt to verify the result in AutoProof failed to prove one temporal property, which on further inspection seemed to be wrong in the original published model, even though the published work claimed to have verified an Abstract State Machine implementation of that model. Correcting the CTL specification to reflect the apparent informal attempt, re-translating again to contracted Eiffel and re-running the verification leads to success. The LTL-to-contracted-Eiffel process is still ad hoc, and tailored to generate the kind of scheme that the target verification tool (AutoProof) can handle best, rather than the simplest or most elegant scheme. Even with this limitation, the results highlight the need for rigor in the verification process, and (on the positive side) demonstrate that the highly advanced mechanized proof technology developed over several decades for the verification of traditional programs also has the potential of handling the demanding needs of embedded systems and other demanding contemporary developments.

Keywords: seamless requirements; design by contract; autoproof; eiffel; landing gear system

DOI: 10.15514/ISPRAS-2017-29(4)-3

For citation: Naumchev A., Mazzara M., Meyer B., Bruel J.-M., Galinier F., Ebersold S. A contract-based method to specify stimulus-response requirements. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 39-54. DOI: 10.15514/ISPRAS-2017-29(4)-3

1. Overview and main results

The present article describes a technique for specification and verification of stimulus-response requirements using a general-purpose programming language (Eiffel) and a program prover (AutoProof [1]) based on the principles of Design by Contract [2].

Real-time, or reactive, systems are often run by a software controller that repeatedly executes one and the same routine and it is specified to take actions at specific time intervals or according to external stimuli [3]. This architecture is reasonable when the software has to react timely to non-deterministic changes in the environment. In this case the program should react to the external stimuli in small steps, so that in the event of a new change it responds timely.

Computation tree logics (CTL) [4] represent a frequent choice when it comes to capturing stimulus-response requirements. Although it may be easier to reason about requirements using declarative logic like CTL, the reasoning may be of little value for the software developer who will implement the requirements. Mainstream programming languages are all imperative, and the translation between declarative requirements and imperative programs is semi-formal.

Requirements have to be of imperative nature from the beginning. This would bridge the gap in how customers and developers understand them. For a software developer it is preferable to reason about the future program without switching to an additional formalism, notation and tools not connected to the original programming language and the IDE.

The present article describes a technique to achieve this goal, in particular:

- Introduces the Landing Gear System (LGS) case study and the LGS baseline requirements (Section 2).
- Generalizes the LGS baseline requirements, maps them to a well-established taxonomy, and complements the taxonomy (Section 3).
- Provides a general scheme for capturing semantics of the stimulus-response requirements in the form of imperative program routines with assertions (Section 4).
- Exercises utility of the approach by applying it to an Abstract State Machine (ASM) specification of the Landing Gear System case study (Section 5).
- Concludes the possibility of statically checking a sequential imperative program directly against a stimulus-response requirement whose semantics

is expressed in the same programming language through conditionals, loops, and assertions (Section 7).

Application of the technique leads to discovery of an error in the published model of the LGS ASM [5]. The error is not present in the specification the authors have actually used for proving the properties, but the error has found its way into the publication.

2. The landing gear system

Landing Gear System was proposed as a benchmark for techniques and tools dedicated to the verification of behavioral properties of systems [6]. It physically consists of the landing set, a gear box that stores the gear in the retracted position, and a door attached to the box (Figure 1). The door and the gear are actuated independently by a digital controller. The controller reacts to changes in position of a handle in the cockpit by initiating either gear extension or retraction process. The task is to program the controller so that it correctly aligns in time the events of changing the handle's position and sending commands to the door and the gear actuators.

3. Stimulus-response requirements

The LGS case study defines a number of requirements, including several for the normal mode of operation (Figure 2). The requirements communicate a common meaning of the form: *If **stimulus** holds, then **response** will eventually hold in the future*. For requirement $R_{11}bis$,
$stimulus \Leftrightarrow "The\ operation\ mode\ is\ normal\ and\ the\ handle\ is\ DOWN"$ and
$response \Leftrightarrow (stimulus \Rightarrow "The\ gear\ is\ down\ and\ the\ door\ is\ closed")$.
The implication in the definition of *response* reflects the "and stays DOWN" part of the original requirement. In addition to that, requirements R_{21} and R_{22} communicate something else:

- Once *response* holds in the presence of *stimulus*, and *stimulus* holds forever, *response* will hold forever.

3.1 Temporal interpretation of the requirements

The authors of the LGS ASM specification start with a ground model that satisfies a subset of requirements, and then refine the model to satisfy more requirements. The present article focuses on their ground model and the corresponding baseline requirements it covers (Figure 2). The work expresses the baseline requirements as CTL properties. The CTL interpretation assigns precise meanings to the requirements by assuming small-step execution semantics of ASM's. In particular, for requirements $R_{11}bis$ and $R_{12}bis$ "the future" means "after a finite number of execution steps", while for R_{21} and R_{22} "the future" means "after one execution step".

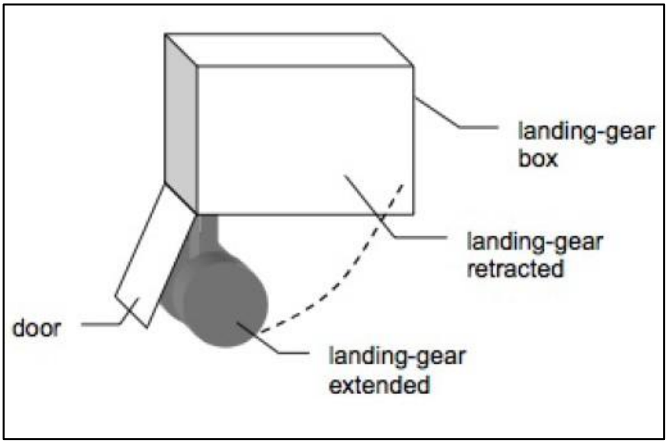


Fig. 1. Landing set (source: [6]).

(R_{11}*bis*)	When the command line is working (normal mode), if the landing gear command handle has been pushed DOWN and stays DOWN, then eventually the gears will be locked down and the doors will be seen closed.
(R_{12}*bis*)	When the command line is working (normal mode), if the landing gear command handle has been pushed UP and stays UP, then eventually the gears will be locked retracted and the doors will be seen closed.
(R_{21})	When the command line is working (normal mode), if the landing gear command handle remains in the DOWN position, then retraction sequence is not observed.
(R_{22})	When the command line is working (normal mode), if the landing gear command handle remains in the UP position, then outgoing sequence is not observed.

Fig. 2. Baseline LGS requirements.

The finite number of steps in R_{11}*bis* and R_{12}*bis* may be unacceptably large though for a system like an LGS of an aircraft. In particular, flights have some expected durations, and the gears have to react to commands in some limited time frame as well. The following two major categories of stimulus-response requirements stem from the speculations above:

- *If stimulus holds, then response will hold in not more than k execution steps.*

Requirements of this form are also called maximal distance requirements [7].

- *If stimulus holds, then response will hold in exactly k execution steps.* Requirements of this form are also called exact distance, or delay requirements.

These two categories are not enough though for capturing stimulus-response requirements. For example, if according to $R_{11}bis$ the gears are locked down and the doors seen closed as the result of the handle staying down, we want this state to be stable if the handle stays down. This leads us to stimulusresponse requirements of the following form:

- *If response holds under stimulus, it will still hold after one execution step in the presence of that stimulus.* Let us call such requirements response stability requirements.

It makes sense to complement requirements ($R_{11}bis$) and ($R_{12}bis$) with the corresponding response stability requirements (Figure 3): not only do we want the LGS to respond to a change in the handle's position, but we also want it to maintain the response if the position does not change.

($R_{11}rs$)	If the gears are locked extended and the doors are closed when the landing gear command handle is DOWN, this state will still hold if the handle stays DOWN.
($R_{12}rs$)	If the gears are locked retracted and the doors are closed when the landing gear command handle is UP, this state will still hold if the handle stays UP.

Fig. 3. LGS response stability requirements.

4. Translation of stimulus-response requirements

Assuming the presence of an infinite loop **from until False loop** main **end** that runs a reactive system, a temporal stimulus-response requirement (Section 3.1) takes the form of a routine with an assertion (**check end** construct in Eiffel). The authors draw this idea from the notion of a specification driver [8] - a contracted routine that forms a proof obligation in Hoare logic. AutoProof is a prover of Eiffel programs that makes it possible to statically check the assertions.

```
response_holds_within_k_steps
-- If stimulus holds, response will hold within k steps.
 local
  steps: NATURAL
 do
```

```
        if (stimulus) then
          from
            steps := 0
          until
            response or (steps = k)
          loop
            main
            steps := steps + 1
          end
          check response end
        end
      end
```

Fig. 4. Representation of a maximal distance requirement. Regardless of the actual reason for the loop to terminate, the response has to hold if the stimulus held at the entry to the loop.

```
    response_holds_in_k_steps
    -- If stimulus holds, response will hold in k steps.
      local
        steps: NATURAL
      do
        if (stimulus) then
          from
            steps := 0
          until
            response or (steps=k)
          loop
            main
            steps := steps + 1
          end
          check (response and (steps = k)) end
        end
      end
```

Fig. 5. Representation of an exact distance requirement. Both of the loop exit conditions have to hold for the first time simultaneously if the stimulus held at the entry to the loop.

4.1 Maximal distance

In the representation of a maximal distance requirement (Figure 4) the "**if** stimulus **then**" clause captures the presence of the stimulus before the up-to-*k*-length execution fragment, and the "**check** response **end**" assertion expresses the need for the response upon completion of the subexecution. The sub-execution may complete for two possible reasons: either occurrence of the response or consumption of all of the available *k* steps. In the both cases the response has to hold.

4.2 Exact distance

Representation of an exact distance requirement (Figure 5) is very similar to that one of a maximal distance, with the "**check** (response **and** (steps = k)) **end**" assertion that makes the difference. Regardless of whether the loop terminates because of "response **or** steps = k", the both have to hold upon the termination.

4.3 Response stability

Representation of a response stability requirement (Figure 6) says: whenever response holds under stimulus in a state, it will still hold in the presence of the same stimulus in the next state.

```
response_is_stable_under_stimulus
-- response keeps holding under stimulus.
  do
    if (stimulus and response) then
      main
      check (stimulus implies response) end
    end
  end
```

Fig. 6. Representation of a response stability requirement. If response holds under stimulus in some state, the response should hold in the next state in the presence of the same stimulus.

5. Applying the translation scheme to the landing gear example

The article exercises the approach on the LGS ASM specification, which is operational by the definition and thus is a subject for translation into an imperative program. For this reason the present section starts with explanation of the rules according to which the authors converted the original specification into an Eiffel program.

5.1 Translation of ASM specifications

An ASM specification is a collection of rules taking one of the following three forms [9]: assignment (Section V-A1), do-in-parallel (Section V-A2), and conditional (Section V-A3). If we have general rules for translating these operators into Eiffel then we will be able to translate an arbitrary ASM into an Eiffel program.

An ASM *assignment* looks as follows:

$$f\ (t_1, \ldots, t_j) := t_0 \qquad (1)$$

The semantics is: update the current content of location $\lambda = (f, (a_1, .., a_j))$, where a_i are values referenced by t_i, with the value referenced by t_0.

In Eiffel locations are represented with class attributes, so an ASM's location update corresponds in Eiffel to an attribute assignment.

An ASM *do-in-parallel* operation can apply several rules simultaneously in one step:

$$R_1 || \ldots || R_k \qquad (2)$$

In order to emulate a parallel assignment in a synchronous setting, one needs to assign first to fresh variables and then assign their values to the original ones. For example, an ASM do-in-parallel statement

$$a, b := \max(a - b, b), \min(a - b, b) \qquad (3)$$

in Eiffel would look like:

```
local
  a_intermediate, b_intermediate: INTEGER
do
  a_intermediate := max (a−b, b)
  b_intermediate := min (a−b, b)
  a := a_intermediate
  b := b_intermediate
end
```

An attempt to update in parallel identical locations in an ASM corresponds semantically to a crash. The translation scheme not only preserves but strengthens this semantics: an Eiffel program with two local variables declared with identical names will not compile.

Conditional: An ASM conditional **if** t **then** R1 **else** R2 carries the same meaning as in Eiffel, so the translation is straightforward.

5.2 An error in the ground model

Translation of the original LGS ASM specification into Eiffel is publicly available in a GitHub repository [10] The error is not handling the situation when the door is closing and the handle is pushed down, in which case the ground model will not meet requirement ($R_{11}bis$). To catch this error with the SVR method one needs first to introduce it back by commenting out two lines in the “open_door” routine of the Eiffel translation:

```
open_door
 do
  inspect door_status
  when closed_position then
   door_status := opening_state
  -- when closing_state then
  --   door_status := opening_state
```

and then submit routine r11_bis to verification with AutoProof; the verification will fail. We have contacted an author of the article that contains the erroneous ASM specification, and he admitted the presence of the error.

5.3 Requirements

The two classes include the translations of the baseline requirements plus the response stability requirements introduced in the present article. We do not discuss all of them here: requirements ($R_{11}bis$) and ($R_{12}bis$), (R_{21}) and (R_{22}), ($R_{11}rs$) and ($R_{12}rs$) are pairwise similar, which is why we prefer to pick one from each pair.

Translation of requirement r11_bis (Figure 7) is an application of the response_holds_within_k_steps pattern (Figure 4), where:

- stimulus equates to:
  is_normal_mode **and** (handle_status = is_handle_down)
- response equates to:
  (**not** (is_normal_mode **and** (handle_status = is_handle_down))) **or** ((gear_status = is_gear_extended) **and** (door_status = is_door_closed))

The idea behind the response is that there may be two reasons for the gear not to extend and the door not to close:

- An abnormal situation that leads to quitting the normal mode.
- The crew changes their mind and pushes the handle up.

```
r11_bis
-- If (is_normal_mode and (handle_status = is_handle_down)) hold and remain,
-- ((gear_status = is_gear_extended) and (door_status = is_door_closed)) will hold within 10 steps.
```

```
  local
    steps: NATURAL
  do
    if (is_normal_mode and (handle_status = is_handle_down)) then
      from
        steps := 0
      until
        (not (is_normal_mode and (handle_status = is_handle_down)))
or ((gear_status = is_gear_extended) and
        (door_status = is_door_closed)) or (steps = 10)
      loop
        main
        steps := steps + 1
      end
      check (not (is_normal_mode and (handle_status =
is_handle_down))) or
        ((gear_status = is_gear_extended) and (door_status =
is_door_closed)) end
     end
  end
```

Fig. 7. Translation of the "r11 bis" requirement.

```
r21
-- If (is_normal_mode and (handle_status = is_handle_up)) holds and
remains,
-- (gear_status= is_gear_extending) will hold within 1 step.
  local
    steps: NATURAL
  do
    if (is_normal_mode and (handle_status = is_handle_up)) then
      from
        steps := 0
      until
        (not (is_normal_mode and (handle_status = is_handle_up))) or
        (gear_status = is_gear_extending) or
        (steps = 1)
      loop
        main
        steps := steps + 1
      end
      check (not (is_normal_mode and (handle_status = is_handle_up))) or
        (gear_status = is_gear_extending) end
    end
  end
```

Fig. 8. Translation of the "r21" requirement.

```
r11_rs
-- ((gear_status = is_gear_extended) and (door_status = is_door_closed)) keeps holding under
-- (is_normal_mode and (handle_status = is_handle_down))
  do
    if ((is_normal_mode and (handle_status = is_handle_down)) and
        ((gear_status = is_gear_extended) and (door_status = is_door_closed))) then
      main
      check ((is_normal_mode and (handle_status = is_handle_down)) implies
        ((gear_status = is_gear_extended) and (door_status = is_door_closed))) end
    end
  end
```

Fig. 9. Translation of the "r11 rs" requirement.

6. Related work

Modeling of real-time computation and related requirements is a well-investigated matter [12]. Representation of real-time requirements, expressed in general or specific form, is a challenging task that has been attacked by the use of several formalisms both in sequential and concurrent settings, and in a broad set of application domains. The difficulty (or impossibility) to fully represents general real-time requirements other than in natural language, or making use of excessively complicated formalisms (unsuitable for software developers), has been recognized.

In [13] the domain of real-time reconfiguration of system is discussed, emphasizing the necessity of adequate formalisms. The problem of modeling real time in the context of services orchestration in Business Process, and in presence of abnormal behavior has been examined in [14] and [15] by means, respectively, of process algebra and temporal logic. Modeling of protocols also requires real-time aspects to be represented [16]. Event-B has also been used as a vector for real-time extension [17] in order to handle embedded systems requirements.

In all these studies, the necessity emerged of focusing on specific typology of requirements using ad-hoc formalisms and techniques, and making use of abstractions. The notion of "real-time" is often abstracted as *number of steps*, a metric commonly used. In this paper we follow the same approach, inheriting both strength (simplicity of the model and effectiveness for applicative purposes) and limitations (temporal logic and time automata themselves miss to capture a precise notion of *real-time*).

7. Conclusions and future work

Software developers reason in an *imperative/operational* manner. This claim is supported both by anecdotal experience and by empirical evidence [18]. Requirements expressed in imperative/operational fashion would therefore results of easier comprehensions for developers and would simplify the process of negotiation behind requirements elicitation. In the method described in this paper, requirements are expressed in a formalism (or language) that seamlessly stay the same along the whole process, without the need of switching between different instruments or mental paradigms. At the same time, the linguistic tool used to define them also allows for automatic verification of correctness.

The meaning of correctness here remains subject to the assumption that requirements engineers and stakeholders agree on a list of desiderata that is indeed the intended one. Assuming a non-faulty process of intention transferring (and this assumption is common to any other approach too), requirements are now more easily manageable by software engineerings all the way from elicitation to verification.

The result of elicitation process is a set of requirements in natural language. The full realization of the presented method would imply an automatic (or semi-automatic) translation from natural language into a structured representation that, although completely intuitive for software developers, it is possibly not easy to manage for average stakeholders. The first part of this process, i.e., the translation from natural language into the current representation (and back) is under development. A tool automatically translates semi-structured natural language into the Hoare-triple-based representation [19], allowing also the opposite direction, i.e. back to natural language [20], so that software engineers would be able to negotiate back requirements with stakeholders using a format they would comprehend. The role of the requirement engineers would then consist in concluding the elicitation phase with a set of requirements in semi-structured natural language, which the tool would be able to process in an entirely automatic manner.

This paper supports the idea of seamless development describing a method supported by a formalism that stay the same along the whole process, from requirements to deployment. Alternative approaches have also been experimented which make use of formalism-based toolkits, where ad hoc notations are adopted for each development phase [21].

References

[1]. J. Tschannen, C. A. Furia, M. Nordio, and N. Polikarpova, “Autoproof: Auto-active functional verification of object-oriented programs,” *arXiv preprint arXiv:1501.03063*, 2015.

[2]. B. Meyer, *Touch of Class: learning to program well with objects and contracts*. Springer, 2009.

[3]. I. J. Hayes, M. A. Jackson, and C. B. Jones, *Determining the Specification of a Control System from That of Its Environment*, pp. 154–169. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.

[4]. E. Clarke and E. Emerson, “Design and synthesis of synchronization skeletons using branching time temporal logic,” *Logics of programs*, pp. 52–71, 1982.

[5]. P. Arcaini, A. Gargantini, and E. Riccobene, “Modeling and analyzing using asms: the landing gear system case study,” in *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pp. 36–51, Springer, 2014.

[6]. F. Boniol and V. Wiels, “The landing gear system case study,” in *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pp. 1–18, Springer, 2014.

[7]. R. Koymans, “Specifying real-time properties with metric temporal logic,” *Real-time systems*, vol. 2, no. 4, pp. 255– 299, 1990.

[8]. A. Naumchev and B. Meyer, “Complete contracts through specification drivers,” in *2016 10th International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pp. 160–167, July 2016.

[9]. Y. Gurevich, “Sequential abstract-state machines capture sequential algorithms,” *ACM Transactions on Computational Logic (TOCL)*, vol. 1, no. 1, pp. 77–111, 2000.

[10]. A. Naumchev, “Lgs asm ground model in eiffel” https://github.com/anaumchev/lgs_ground_model, 2017.

[11]. N. Polikarpova, J. Tschannen, C. A. Furia, and B. Meyer, “Flexible invariants through semantic collaboration,” in *FM 2014: Formal Methods*, pp. 514–530, Springer, 2014.

[12]. H. Yamada, “Real-time computation and recursive functions not real-time computable,” *IRE Transactions on Electronic Computers*, vol. EC-11, pp. 753–760, Dec 1962.

[13]. M. Mazzara and A. Bhattacharyya, “On modelling and analysis of dynamic reconfiguration of dependable real time systems,” in *Proceedings of the 2010 Third International Conference on Dependability*, DEPEND ’10, (Washington, DC, USA), pp. 173–181, IEEE Computer Society, 2010.

[14]. M. Mazzara, “Timing issues in web services composition,” in *Formal Techniques for Computer Systems and Business Processes, European Performance Engineering Workshop, EPEW 2005 and International Workshop on Web Services and Formal Methods, WS-FM 2005, Versailles, France, September 1-3, 2005, Proceedings*, pp. 287–302, 2005.

[15]. L. Ferrucci, M. M. Bersani, and M. Mazzara, “An LTL semantics of business workflows with recovery,” in ICSOFTPT 2014 - Proceedings of the 9th International Conference on Software Paradigm Trends, Vienna, Austria, 2931 August, 2014, pp. 29–40, 2014.

[16]. M. Berger and K. Honda, “The two-phase commitment protocol in an extended pi-calculus,” Electr. Notes Theor. Comput. Sci., vol. 39, no. 1, pp. 21–46, 2000.

[17]. A. Iliasov, A. Romanovsky, L. Laibinis, E. Troubitsyna, and T. Latvala, “Augmenting event-b modelling with real time verification,” in Proceedings of the First International

Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches, FormSERA ’12, 2012.

[18]. D. Fahland, D. Lubke, J. Mendling, H. Reijers, B. Weber,¨ M. Weidlich, and S. Zugal, Declarative versus Imperative Process Modeling Languages: The Issue of Understandability. Springer Berlin Heidelberg, 2009.

[19]. A. Bormotova, “Translation of natural language into hoare triples.” https://github.com/An-Dole/ Semantic-mapping.

[20]. V. Skukov, “Translation of hoare triples into natural language.” https://github.com/flosca/hybrid.

[21]. R. Gmehlich, K. Grau, F. Loesch, A. Iliasov, M. Jackson, and M. Mazzara, “Towards a formalism-based toolkit for automotive applications,” in 1st FME Workshop on Formal Methods in Software Engineering, FormaliSE 2013, San Francisco, CA, USA, May 25, 2013, pp. 36–42, 2013.

Контрактный метод спецификации реактивных требований

[1]*А. Наумчев <a.naumchev@innopolis.ru>*
[1]*М. Маццара <m.mazzara@innopolis.ru>*
[1, 2, 3]*Б. Мейер <Bertrand.Meyer@inf.ethz.ch>*
[3]*Ж.-М. Брюэль <bruel@irit.fr>*
[3]*Ф. Галинье <galinier@irit.fr>*
[3]*С. Эберсоль <ebersold@irit.fr>*
[1]*Университет Иннополис,*
420500, Российская Федерация, г. Иннополис, ул. Университетская, д. 1.
[2]*Миланский технический университет,*
20133, Италия, г. Милан, Piazza Leonardo da Vinci, 32.
[3]*Университет Тулузы,*
31062, Франция, г. Тулуза, Route de Narbonne, 118.

Аннотация. Верификация многих прикладных систем – в частности, встроенных, - включает в себя процессы, исполняющиеся во времени, для моделирования которых обычно используется временн*а*я логика, линейная (LTL) или ветвящаяся (CTL). Наиболее развитые автоматические доказатели программ, однако, основаны на невременн*ы*х теориях: например, на логике Хоара. Возможно ли все же применение этой развитой технологии верификации к более сложным системам? В качестве шага на пути к положительному ответу, мы разработали схему перевода подмножества LTL спецификаций в объектно-ориентированные программы с контрактами на языке Eiffel, которые являются естественными целями для доказателя программ AutoProof. Мы применили эту схему к опубликованной временн*о*й модели широко используемого реалистичного примера, авиационной системы контроля шасси, являющейся своего рода эталонной задачей для сравнения применимости различных методов спецификации. Верификация переведенной спецификации с помощью AutoProof обнаружила ошибку в одном из временн*ы*х свойств. Углубленное изучение данной ошибки привело к обнаружению ошибки в опубликованной абстрактной машине состояний (ASM), которая реализует переведенную модель; авторы публикации, в свою очередь, заявили об успешной верификации. Корректировка исходной спецификации и перевод результата в Eiffel с контрактами с последующей верификацией привели к успешному результату. Процесс перевода из LTL в Eiffel все еще находится в зачаточном состоянии и оптимизирован для используемого инструмента верификации (AutoProof), поэтому схема перевода не выглядит простой и элегантной. Даже с учетом указанных ограничений полученные результаты демонстрируют потенциал технологии автоматического доказательства традиционных программ в части ее применимости к специфичным проблемам встроенных систем.

Ключевые слова: бесшовные требования; проектирование по контракту; autoproof; эйфель; система контроля шасси

DOI: 10.15514/ISPRAS-2017-29(4)-3

Для цитирования: Наумчев А., Маццара М., Мейер Б., Брюэль Ж.-М., Галинье Ф., Эберсоль С. Контрактный метод спецификации реактивных требований. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 39-54 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-3

Список литературы

[1]. J. Tschannen, C. A. Furia, M. Nordio, and N. Polikarpova, “Autoproof: Auto-active functional verification of object-oriented programs,” *arXiv preprint arXiv:1501.03063*, 2015.

[2]. B. Meyer, *Touch of Class: learning to program well with objects and contracts*. Springer, 2009.

[3]. I. J. Hayes, M. A. Jackson, and C. B. Jones, *Determining the Specification of a Control System from That of Its Environment*, pp. 154–169. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.

[4]. E. Clarke and E. Emerson, “Design and synthesis of synchronization skeletons using branching time temporal logic,” *Logics of programs*, pp. 52–71, 1982.

[5]. P. Arcaini, A. Gargantini, and E. Riccobene, “Modeling and analyzing using asms: the landing gear system case study,” in *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pp. 36–51, Springer, 2014.

[6]. F. Boniol and V. Wiels, “The landing gear system case study,” in *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pp. 1–18, Springer, 2014.

[7]. R. Koymans, “Specifying real-time properties with metric temporal logic,” *Real-time systems*, vol. 2, no. 4, pp. 255– 299, 1990.

[8]. A. Naumchev and B. Meyer, “Complete contracts through specification drivers,” in *2016 10th International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pp. 160–167, July 2016.

[9]. Y. Gurevich, “Sequential abstract-state machines capture sequential algorithms,” *ACM Transactions on Computational Logic (TOCL)*, vol. 1, no. 1, pp. 77–111, 2000.

[10]. A. Naumchev, “Lgs asm ground model in eiffel..” https://github.com/anaumchev/lgs_ground_model, 2017.

[11]. N. Polikarpova, J. Tschannen, C. A. Furia, and B. Meyer, "Flexible invariants through semantic collaboration," in *FM 2014: Formal Methods*, pp. 514–530, Springer, 2014.

[12]. H. Yamada, "Real-time computation and recursive functions not real-time computable," *IRE Transactions on Electronic Computers*, vol. EC-11, pp. 753–760, Dec 1962.

[13]. M. Mazzara and A. Bhattacharyya, "On modelling and analysis of dynamic reconfiguration of dependable real time systems," in *Proceedings of the 2010 Third International Conference on Dependability*, DEPEND '10, (Washington, DC, USA), pp. 173–181, IEEE Computer Society, 2010.

[14]. M. Mazzara, "Timing issues in web services composition," in *Formal Techniques for Computer Systems and Business Processes, European Performance Engineering Workshop, EPEW 2005 and International Workshop on Web Services and Formal Methods, WS-FM 2005, Versailles, France, September 1-3, 2005, Proceedings*, pp. 287–302, 2005.

[15]. L. Ferrucci, M. M. Bersani, and M. Mazzara, "An LTL semantics of business workflows with recovery," in ICSOFTPT 2014 - Proceedings of the 9th International Conference on Software Paradigm Trends, Vienna, Austria, 2931 August, 2014, pp. 29–40, 2014.

[16]. M. Berger and K. Honda, "The two-phase commitment protocol in an extended pi-calculus," Electr. Notes Theor. Comput. Sci., vol. 39, no. 1, pp. 21–46, 2000.

[17]. A. Iliasov, A. Romanovsky, L. Laibinis, E. Troubitsyna, and T. Latvala, "Augmenting event-b modelling with real time verification," in Proceedings of the First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches, FormSERA '12, 2012.

[18]. D. Fahland, D. Lubke, J. Mendling, H. Reijers, B. Weber,¨ M. Weidlich, and S. Zugal, Declarative versus Imperative Process Modeling Languages: The Issue of Understandability. Springer Berlin Heidelberg, 2009.

[19]. A. Bormotova, "Translation of natural language into hoare triples." https://github.com/An-Dole/ Semantic-mapping.

[20]. V. Skukov, "Translation of hoare triples into natural language." https://github.com/flosca/hybrid.

[21]. R. Gmehlich, K. Grau, F. Loesch, A. Iliasov, M. Jackson, and M. Mazzara, "Towards a formalism-based toolkit for automotive applications," in 1st FME Workshop on Formal Methods in Software Engineering, FormaliSE 2013, San Francisco, CA, USA, May 25, 2013, pp. 36–42, 2013.

Fast L^1 Gauss Transforms for Edge-Aware Image Filtering

[1,2]Dina Bashkirova <dina.bashkirova@riken.jp>
[1]Shin Yoshizawa <shin@riken.jp>
[2]Roustam Latypov <roustam.latypov@kpfu.ru>
[1]Hideo Yokota <hyokota@riken.jp>
[1]Image Processing Research Team,
RIKEN Center for Advanced Photonics, RIKEN,
2-1, Hirosawa, Wako, Saitama, 351-0198, Japan
[2]Institute of Computational Mathematics and Information Technologies,
Kazan Federal University,
35 Kremlyovskaya, Kazan, Russia, 420008

Abstract. Gaussian convolution and its discrete analogue, Gauss transform, have many science and engineering applications, such as mathematical statistics, thermodynamics and machine learning, and are widely applied to computer vision and image processing tasks. Due to its computational expense (quadratic and exponential complexities with respect to the number of points and dimensionality, respectively) and rapid spreading of high quality data (bit depth/dynamic range), accurate approximation has become important in practice compared with conventional fast methods, such as recursive or box kernel methods. In this paper, we propose a novel approximation method for fast Gaussian convolution of two-dimensional uniform point sets, such as 2D images. Our method employs L1 distance metric for Gaussian function and domain splitting approach to achieve fast computation (linear computational complexity) while preserving high accuracy. Our numerical experiments show the advantages over conventional methods in terms of speed and precision. We also introduce a novel and effective joint image filtering approach based on the proposed method, and demonstrate its capability on edge-aware smoothing and detail enhancement. The experiments show that filters based on the proposed L1 Gauss transform give higher quality of the result and are faster than the original filters that use box kernel for Gaussian convolution approximation.

Keywords: Gaussian smoothing, Laplace distribution, fast approximation algorithms.

DOI: 10.15514/ISPRAS-2017-29(4)-4

For citation: Bashkirova D.R., Yoshizawa S., Latypov R.H., Yokota H. Fast L1 Gauss Transforms for Edge-Aware Filtering. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 55-72. DOI: 10.15514/ISPRAS-2017-29(4)-4

1. Introduction

Gaussian convolution is a core tool in mathematics and many related research areas, such as probability theory, physics, and signal processing. Gauss transform is a discrete analogue to the Gaussian convolution, and has been widely used for many applications including kernel density estimation [1] and image filtering [2]. Despite its reliable performance and solid theoretical foundations, Gauss transform in its exact form along with other kernel-based methods has a drawback – it is very computationally expensive (has quadratic computational complexity w.r.t. the number of points) and hard to scale to higher dimensions. Which is why there have been many attempts to overcome these problems by creating approximation algorithms, such as fast Gauss transform [3], dualtree fast Gauss transforms [4], fast KDE [5], and Gaussian kd-trees [6]. Also, box kernel averaging [7] and recursive filtering [8] have been popular in computer graphics and image processing because of their simplicity, see the surveys [9], [10] for numerical comparisons of these approximation methods.

Since high bit depth (also dynamic range) images have become popular in both digital entertainment and scientific/engineering applications, it is very important to acquire high approximation precision and to reduce artefacts caused by drastic truncation employed in many conventional methods focused on computational speed. One of the highly accurate methods is called fast L^1 Gauss transform approximation [11] based on using L^1 distance instead of conventional L^2 Euclidean metric. This L^1 metric preserves most of the properties of the L^2 Gaussian, and is separable, hence it allows to perform computations along each dimension separately, which is very beneficial in terms of computational complexity. Also, L^1 Gaussian has only one peak in Fourier domain at the coordinate origin, and therefore its convolution does not have some undesirable artefacts that box kernels and truncation methods usually have. However, this algorithm works only on one-dimensional (1D) point sets, although it can be extended to uniformly distributed points in higher dimensions by performing it separately in each dimension. In order to be able to acquire Gauss transform for non-uniformly distributed two-dimensional points and to further generalize it to higher dimensional cases, we need to extend existing method [11] to the 2D uniform case.

In this paper we propose a novel approximation method for fast Gauss two-dimensional (2D) image transform. Our method is based on extending the fast L^1 Gauss transform approximation on uniformly distributed 2D points that allows to perform Gaussian convolution quickly while preserving high accuracy. We demonstrate that efficiency of the proposed method in terms of computational complexity, numerical timing, and approximation precision.

We also successfully applied our method in the novel filtering approach based on combining the approximated L^1 Gauss transformations into the so-called guided filter [12] (joint image filtering via ridge regression). Our approach reduces computational costs while providing higher quality results compared to the conventional one. We show the application to edge-aware smoothing and image detail enhancement.

2. Fast L1 Gauss Transform

In this section, we briefly describe the 1D domain splitting algorithm [11] employed for fast L^1 Gauss transforms.

Consider the ordered point set $\mathbb{X} = \{x_i\}_{i=1}^N$, $x_i \in \mathbb{R}$, $x_i \geq x_{i-1}$, $\forall i = \overline{2, N}$. Each point x_i has a corresponding value $I_i \in \mathbb{R}$, e.g. pixel intensity in case of images. The L^1 Gauss transform for each point in set $\mathbb{X}$ is given by

$$J(x_j) = \sum_{i=1}^{N} G(x_j - x_i) I_i, \quad G(x) = \exp(-\frac{|x|}{\sigma}), \quad (1)$$

where $G(x)$, $x \in$ R, is a L^1 Gaussian function (also called Laplace distribution in statistics) with its standard deviation σ. It is convenient to decompose L^1 norm by splitting its domain by using the point x_1 such that

$$|x_j - x_i| = \begin{cases} |x_j - x_1| - |x_i - x_1| & \text{if } x_1 \leq x_i \leq x_j, \\ |x_i - x_1| - |x_j - x_1| & \text{if } x_1 \leq x_j \leq x_i. \end{cases} \quad (2)$$

Thus, Gauss transform (1) using the equation (2) becomes

$$J(x_j) = I_i + G(x_j - x_1) \sum_{i=1}^{j-1} \frac{I_i}{G(x_i - x_1)} + \\ + \frac{1}{G(x_j - x_1)} \sum_{i=j+1}^{N} G(x_i - x_1) I_i. \quad (3)$$

Such representation (3) allows to reduce the amount of computational operations, since values $G(x_j - x_1)$, $\frac{1}{G(x_j - x_1)}$ and the sums $\sum_{j+1}^{N} I_i G(x_i - x_1)$ and $\sum_{i=1}^{j-1} \frac{I_i}{G(x_i - x_1)}$ can be precomputed in linear time. However, using the equation (3) may imply some numerical issues, such as overflow, if the distance between x_1 and x_l, $l \in \{i, j\}$ is relatively large. To avoid such issues, this algorithm introduced certain representative points (poles) $\{\alpha_k \in \mathbb{R}\}$ instead of using the single point x_1, where the distance between α_k and x_l is smaller than the length that causes the numerical instability. Hence the equation (3) becomes more complex form, a highly accurate truncation can be applied where $G(\alpha_k - x_j)$ is numerically equal to zero, see [11] for further technical details.

Although this algorithm can be used in case of multidimensional images by applying it separately in each dimension, this separable implementation approach is not applicable to nonuniformly distributed high-dimensional point sets. Therefore, we present a novel and natural extension of the domain splitting concept on 2D cases (images) in the following sections.

3. Two-Dimensional Algorithm

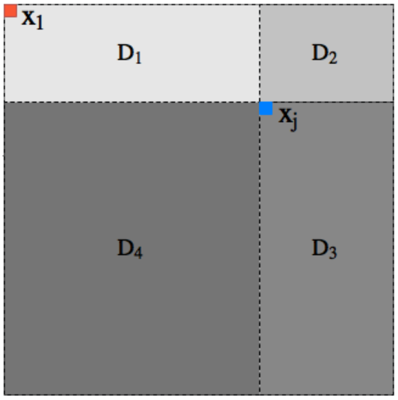
For a given 2D point set $\mathbb{X} = \{\mathbf{x}_i\}_{i=1}^N$, $\mathbf{x}_i = (x_i, y_i) \in \mathbb{R}^2$, L^1 distance between two points in $\mathbb{R}^2$ is given by $|\mathbf{x}_j - \mathbf{x}_i| = |x_j - x_i| + |y_j - y_i|$, thus the Gauss transform (1) is represented by the formula:

$$J(\mathbf{x}_j) = \sum_{i=1}^{N} \exp\left(-\frac{|x_j - x_i| + |y_j - y_i|}{\sigma}\right) I_i$$

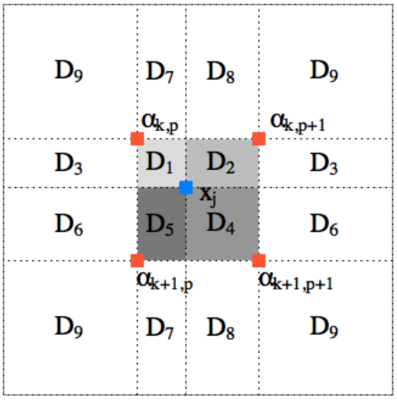
Domain splitting (2) for 2D points is given by

$$|x_j - x_i| + |y_j - y_i| = \begin{cases} |x_j - x_1| - |x_i - x_1| + |y_j - y_1| - |y_i - y_1| & \text{if } \mathbf{x}_i \in D_1 \\ |x_i - x_1| - |x_j - x_1| + |y_j - y_1| - |y_i - y_1| & \text{if } \mathbf{x}_i \in D_2 \\ |x_j - x_1| - |x_i - x_1| + |y_i - y_1| - |y_j - y_1| & \text{if } \mathbf{x}_i \in D_3 \\ |x_i - x_1| - |x_j - x_1| + |y_i - y_1| - |y_j - y_1| & \text{if } \mathbf{x}_i \in D_4, \end{cases}$$

see Fig. 1a for geometric illustration of the domains.



(a) Single pole $\mathbf{x}_1$ case



(b) Multipole $\{\alpha_k\}$ case

Fig. 1. Illustration of 2D domain splliting.

Using the above decomposition, Gauss transform is represented similar to (3):

$$J(\mathbf{x}_j) = I(\mathbf{x}_j) + F(x_j)F(y_j) \sum_{\mathbf{x}_i \in D_1(j)} \frac{1}{F(x_i)F(y_i)} I(\mathbf{x}_i) +$$
$$\frac{F(x_j)}{F(y_j)} \sum_{\mathbf{x}_i \in D_2(j)} \frac{F(y_i)}{F(x_i)} I(\mathbf{x}_i) + \frac{F(y_j)}{F(x_j)} \sum_{\mathbf{x}_i \in D_3(j)} \frac{F(x_i)}{F(y_i)} I(\mathbf{x}_i) +$$
$$\frac{1}{F(x_j)F(y_j)} \sum_{\mathbf{x}_i \in D_4(j)} F(x_i)F(y_i)I(\mathbf{x}_i), \qquad (4)$$

where $F(x_j) \equiv G(x_j - x_1)$ and $F(y_j) \equiv G(y_j - y_1)$.

Precomputation and storage of values $\frac{F(y_j)}{F(x_j)}$ and $\frac{1}{F(x_j)F(y_j)}$ require $O(4N)$ operations and $O(4N)$ space, and all the subsequent sums $F(x_j)F(y_j)$, $\frac{F(x_j)}{F(y_j)}$ can be iteratively computed in $O(N)$ operations. Gauss transform for all points using the formula (4) requires $O(10N)$ as opposed to employing the separable implementation of equation

(3) for $O(6N)$ operations. Since computing the Gauss transform using the equation (4) is numerically troublesome, it is reasonable to divide the space into smaller groups and perform computations separately, as it was proposed in [11]. Let us introduce a novel 2D multipole approach for solving this problem.

Consider a set of poles $\{\alpha_k\}_{k=1}^{M}$, $\alpha_k = (a_k, b_k) \in \mathbb{R}^2$. The distance between points using poles α_k is given by

$$|\mathbf{x}_i - \mathbf{x}_j| = \begin{cases} |x_i - a_k| - |x_j - a_k| + |y_i - b_k| - |y_j - b_k| & \text{if } \mathbf{x}_i \in D_1 \\ |x_j - a_k| - |x_i - a_k| + |y_i - b_k| - |y_j - b_k| & \text{if } \mathbf{x}_i \in D_2 \\ |x_i - a_k| + |x_j - a_k| + |y_i - b_k| - |y_j - b_k| & \text{if } \mathbf{x}_i \in D_3 \\ |x_i - a_k| - |x_j - a_k| + |y_j - b_k| - |y_i - b_k| & \text{if } \mathbf{x}_i \in D_4 \\ |x_j - a_k| - |x_i - a_k| + |y_j - b_k| - |y_i - b_k| & \text{if } \mathbf{x}_i \in D_5 \\ |x_i - a_k| + |x_j - a_k| + |y_j - b_k| - |y_i - b_k| & \text{if } \mathbf{x}_i \in D_6 \\ |x_i - a_k| - |x_j - a_k| + |y_i - b_k| + |y_j - b_k| & \text{if } \mathbf{x}_i \in D_7 \\ |x_j - a_k| - |x_i - a_k| + |y_i - b_k| + |y_j - b_k| & \text{if } \mathbf{x}_i \in D_8 \\ |x_i - a_k| + |x_j - a_k| + |y_i - b_k| + |y_j - b_k| & \text{if } \mathbf{x}_i \in D_9, \end{cases}$$

where

$$D_1 = \{\mathbf{x}_i | x_i \in D_1^x, y_i \in D_1^y\}, D_2 = \{\mathbf{x}_i | x_i \in D_2^x, y_i \in D_1^y\},$$

$$D_3 = \{\mathbf{x}_i | x_i \in D_3^x, y_i \in D_1^y\}, D_4 = \{\mathbf{x}_i | x_i \in D_1^x, y_i \in D_2^y\},$$

$$D_5 = \{\mathbf{x}_i | x_i \in D_2^x, y_i \in D_2^y\}, D_6 = \{\mathbf{x}_i | x_i \in D_3^x, y_i \in D_2^y\},$$

$$D_7 = \{\mathbf{x}_i | x_i \in D_1^x, y_i \in D_3^y\}, D_8 = \{\mathbf{x}_i | x_i \in D_2^x, y_i \in D_3^y\},$$

$$D_9 = \{\mathbf{x}_i | x_i \in D_3^x, y_i \in D_3^y\},$$

$$\begin{aligned} D_1^x &= \{x_i | a_k \le x_i \le x_j \text{ or } x_j \le x_i \le a_k\}, \\ D_2^x &= \{x_i | a_k \le x_j \le x_i \text{ or } x_i \le x_j \le a_k\}, \\ D_3^x &= \{x_i | x_i \le a_k \le x_j \text{ or } x_j \le a_k \le x_i\}, \\ D_1^y &= \{y_i | b_k \le y_i \le y_j \text{ or } y_j \le y_i \le b_k\}, \\ D_2^y &= \{y_i | b_k \le y_j \le y_i \text{ or } y_i \le y_j \le b_k\}, \\ D_3^y &= \{y_i | y_i \le b_k \le y_j \text{ or } y_j \le b_k \le y_i\}, \end{aligned}$$

see Fig. 1b for geometric illustration of the domains with their poles. The point $\mathbf{x}_j$ is assigned for one representative pole defined by

$$\alpha_k(\mathbf{x}_j) = \max_k \{\alpha_k | a_k \le x_j, b_k \le y_j\},$$

which is the closest pole to $\mathbf{x}_j$ that has absolute values of coordinate smaller than $\mathbf{x}_j$.

For each point $\mathbf{x}_j$, the multipole L^1 Gauss transform is given by the equation (5),

$$J(\mathbf{x}_j) = I_j + \mathcal{G}(x_j)\mathcal{G}(y_j) \sum_{\mathbf{x}_i \in D_1} \frac{I_i}{\mathcal{G}(x_i)\mathcal{G}(y_i)} + \frac{1}{\mathcal{G}(x_j)\mathcal{G}(y_j)} \sum_{\mathbf{x}_i \in D_5} \mathcal{G}(x_i)\mathcal{G}(y_i) I_i + \frac{\mathcal{G}(y_j)}{\mathcal{G}(x_j)} \sum_{\mathbf{x}_i \in D_2} \frac{\mathcal{G}(x_i)}{\mathcal{G}(y_i)} I_i + \frac{\mathcal{G}(x_j)}{\mathcal{G}(y_j)} \sum_{\mathbf{x}_i \in D_4} \frac{\mathcal{G}(y_i)}{\mathcal{G}(x_i)} I_i +$$
$$+ \sum_{\alpha_k \in D_9} A_k^j + \sum_{\alpha_k \in D_7} B_k^j + \sum_{\alpha_k \in D_8} C_k^j + \sum_{\alpha_k \in D_3} D_k^j + \sum_{\alpha_k \in D_6} E_k^j, \tag{5}$$
$$A_k^j = \mathcal{G}(x_j)\mathcal{G}(y_j) \sum_{\mathbf{x}_i = \lambda(k)}^{\lambda(k+1)-1} \mathcal{G}(x_i)\mathcal{G}(y_i) I_i, \qquad B_k^j = \mathcal{G}(x_j)\mathcal{G}(y_j) \sum_{\mathbf{x}_i = \lambda(k)}^{\lambda(k+1)-1} \frac{\mathcal{G}(y_i)}{\mathcal{G}(x_i)} I_i, \qquad C_k^j = \frac{\mathcal{G}(y_j)}{\mathcal{G}(x_j)} \sum_{\mathbf{x}_i = \lambda(k)}^{\lambda(k+1)-1} \mathcal{G}(x_i)\mathcal{G}(y_i) I_i,$$
$$D_k^j = \mathcal{G}(x_j)\mathcal{G}(y_j) \sum_{\mathbf{x}_i = \lambda(k)}^{\lambda(k+1)-1} \frac{\mathcal{G}(x_i)}{\mathcal{G}(y_i)} I_i, \qquad E_k^j = \frac{\mathcal{G}(x_j)}{\mathcal{G}(y_j)} \sum_{\mathbf{x}_i = \lambda(k)}^{\lambda(k+1)-1} \mathcal{G}(x_i)\mathcal{G}(y_i) I_i.$$

where $\mathcal{G}(x_j) \equiv G(x_j - a_k), \mathcal{G}(y_j) \equiv G(y_j - b_k)$,

and $\lambda(\cdot)$ is an index function defined by

$$\lambda(k) = \min_{1 \le j \le N} (\mathbf{x}_j | a_k \le x_j < a_{k+1} \text{ and } b_k \le y_j < b_{k+1}).$$

For the sake of simplicity, we assume that the numbers of poles in 2D are same *M*. Following [11], *M* and the poles $\{\alpha_k\}$ are given by

$$\{a_k\} = \{b_k\} = \frac{\{0, 1, 2, ..., (M-1)\}w}{M}, \tag{6}$$
$$w = \max(|x_1 - x_N|, |y_1 - y_N|), \ M = [\frac{w}{\varphi\sigma \log(\text{MAX})}]$$

where [·] is the ceiling function, MAX is the maximum value of precision (e.g., double floating point: DBL_MAX in C programming language), and ϕ is a user-specified parameter (0.5 is employed in our numerical experiments). The above pole selection scheme leads to $\max(G(a_{k+1} - a_k), G(b_{k+1} - b_k)) < \text{MAX}$ which theoretically guarantees numerical stability in our method.

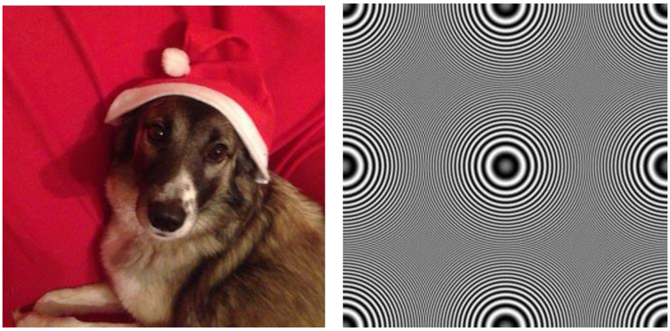
When the distance between poles is determined by the equation (6) and $G(\alpha_k - \mathbf{x}_j)$ becomes numerically zero if $|\alpha_k - \mathbf{x}_j| > \frac{w}{\varphi M}$, we can efficiently truncate Gauss transform by approximating the values:

$$\sum_{\alpha_k \in D_9} A_k^j \approx \sum_{\alpha_k \in \mu(D_9)} A_k^j, \sum_{\alpha_k \in D_7} B_k^j \approx \sum_{\alpha_k \in \mu(D_7)} B_k^j,$$
$$\sum_{\alpha_k \in D_8} C_k^j \approx \sum_{\alpha_k \in \mu(D_8)} C_k^j, \sum_{\alpha_k \in D_3} D_k^j \approx \sum_{\alpha_k \in \mu(D_3)} D_k^j,$$
$$\sum_{\alpha_k \in D_6} E_k^j \approx \sum_{\alpha_k \in \mu(D_6)} E_k^j,$$

where $\mu(D_*) = \{\mathbf{x}_i \in D_* \mid |\alpha_k(\mathbf{x}_j) - \alpha_k(\mathbf{x}_i)| \le \frac{w}{\varphi M}\}$.

In other words, instead of computing terms $A_k^j, B_k^j, C_k^j, D_k^j, E_k^j$ across all the corresponding point sets, we consider only the neighbouring points, which allows to avoid nested loop structures in our implementation and speed up the computational process.

As in the 1D algorithm [11], the terms can be iteratively computed in linear time. Assume that an image consists of $\sqrt{N} \times \sqrt{N}$ pixels and the number of poles along each dimension is *M*, total complexity of our method is $O(16N + 2\frac{\sqrt{N}}{M} + 4\frac{N}{M^2})$ which is a little bit slower than the separable implementation employed in [11] that requires $O(12N + 2\sqrt{N} + M)$ operations.



(a) Input image 1 *(b) Input image 2*

Fig. 2. Input images.

4. Numerical Experiments

We held all the experiments on Intel Core i7-6600U 2.60 GHz dual core computer with 16GB RAM and a 64-bit operating system. We compared the multipole version of our algorithm with box kernel (Box) using moving average method [7], the 1D domain splitting (YY14) with separable implementations [11], and Fast Discrete Cosine Transform (FDCT) via the FFT package [13] well-known for its efficiency.

To evaluate the performance of the methods mentioned above we used randomly generated 2D point sets with 10 different sizes from 128^2 to 5120^2 and 10 various values of σ = 5,10,...,50. The radius for the Box method was chosen equal to σ. The timing results (see Fig. 5) show that our method is slightly slower than the 1D domain splitting (YY14) despite its theoretical complexity is much larger. It is worth noticing that the implementation of our method can be further improved by using GPU-based or parallel computing techniques.

However, the accuracy evaluation results (see Table 1) show that our method achieves best approximation quality among the discussed methods. We evaluate the precision using $E_{\max}$ and PSNR measures. Consider I^e is the exact result of L^1 Gauss transform, I^a is the approximation achieved by a given algorithm, and $d_i = |I_i^e - I_i^a|$. $E_{\max}$ is calculated using formula

$$E_{\max} = \max_{1 \le i \le N} d_i.$$

We also use peak signal-to-noise ratio (PSNR) [2] to measure the performance of our algorithm according to the equation

$$\text{PSNR} = -10 \log(\sum_{i=1}^{N} (\frac{d_i}{\max(I_i^e, I_i^a)})^2).$$

We performed linear image smoothing by the following normalized convolutions for each color channel:

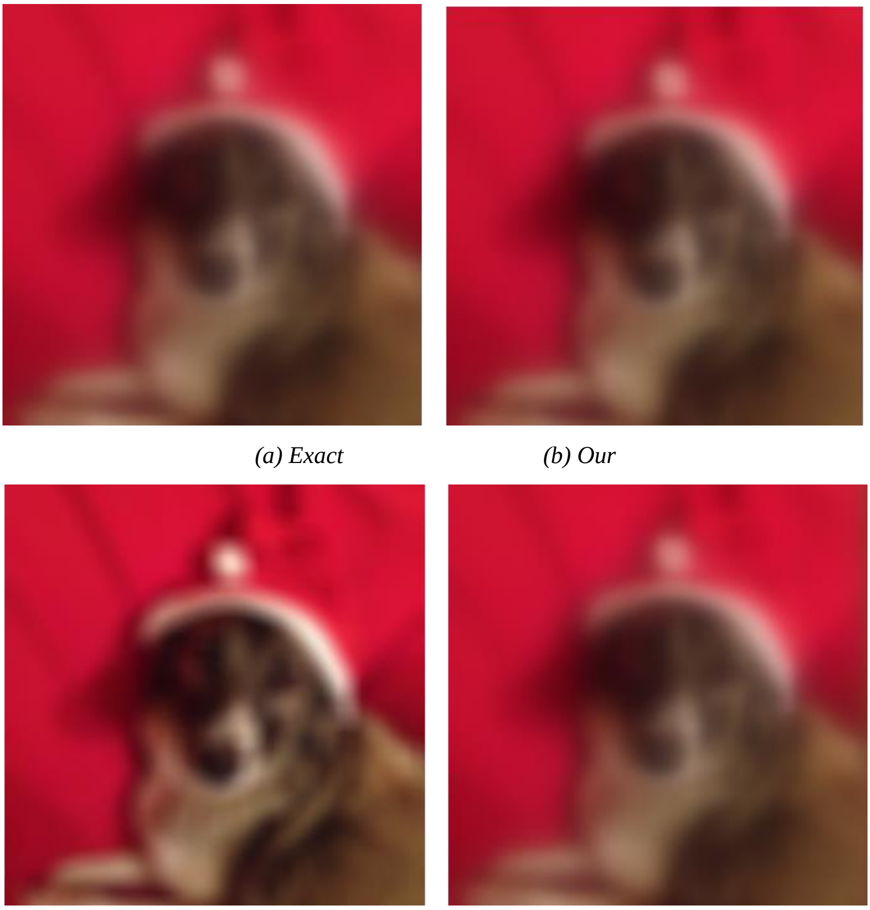
$$\frac{\int G(\mathbf{x}-\mathbf{y})I(\mathbf{y})d\mathbf{y}}{\int G(\mathbf{x}-\mathbf{y})d\mathbf{y}} \rightarrow \frac{J(\mathbf{x}_j)}{\sum_i^N G(\mathbf{x}_j-\mathbf{x}_i)}$$

where the denominator is also obtained by our method convolving L^1 Gaussian with the image whose intensity is equal to one everywhere.

Fig. 3 illustrates the smoothing results using naive implementation (Exact), our method, Box kernel, and FDCT algorithms. The gradient magnitude $|\nabla I|$ of smoothed images on Figs. 4 and 6 show that, in contrast to FDCT and box kernel, our method does not produce some undesirable artifacts and is extremely close to the exact implementation.

Table 1. Precision and speed evaluation results (speed measured in Mpix/sec).

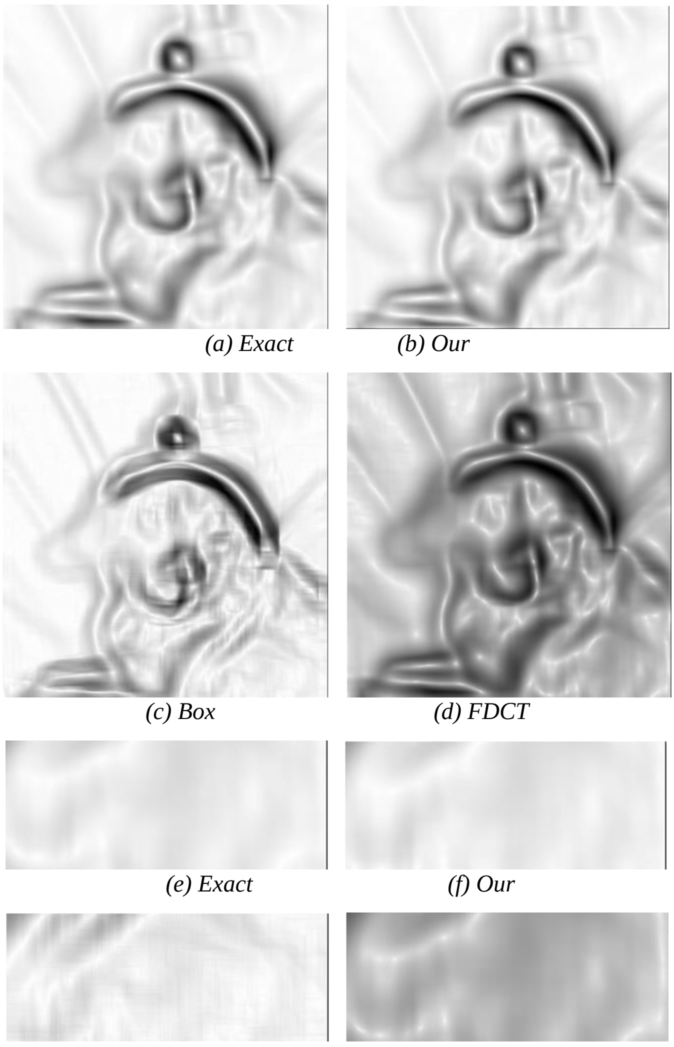
	Our	YY14	FDCT	Box
*E*max	**1.8×10^{-11}**	3.8×10^{-10}	0.44	3.73
PSNR	**291.05**	281.81	58.98	41.45
Speed	7.19	**9.76**	3.37	8.58



(a) Exact *(b) Our*

(c) Box *(d) FDCT*

Fig. 3. Results of smoothing (σ = 20), where the input image is given by Fig.2a.



(a) Exact *(b) Our*

(c) Box *(d) FDCT*

(e) Exact *(f) Our*

(g) Box *(h) FDCT*

Fig. 4: Visualisation of $|\nabla I|$ for comparison of artifacts ($\sigma = 20$).

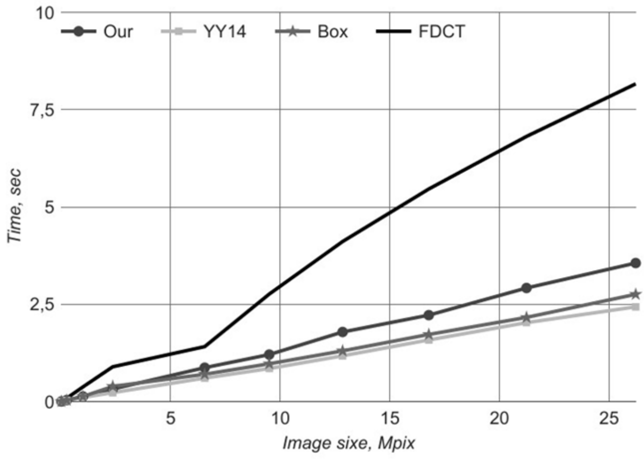
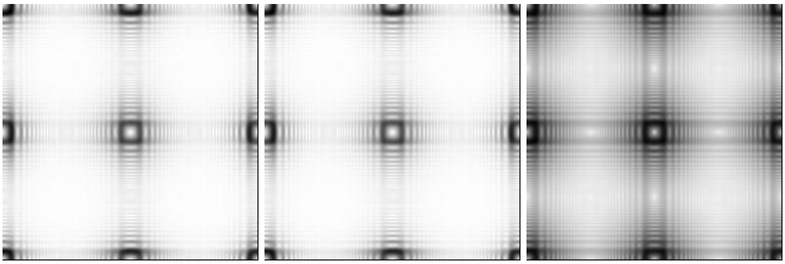
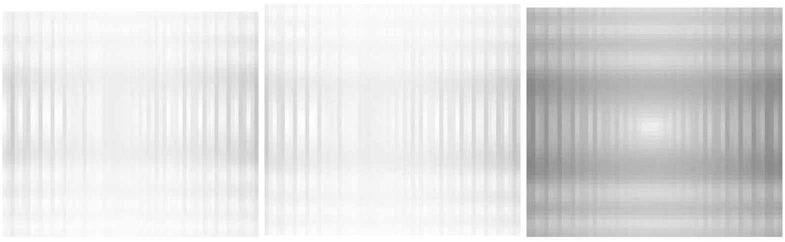


Fig. 5: Timing with respect to image size (averaged by σ).



(a) Exact *(b) Our* *(c) FDCT*



(a) Exact *(b) Our* *(c) FDCT*

Fig. 6: Visualisation of $|\nabla I|$ for comparison of artifacts of FDCT (σ = 20), where the input image is given by Fig.2b.

5. Edge-Aware Filtering

The proposed algorithm for Gauss transform approach can be applied in various computer vision tasks. We present one of the possible applications of our method by introducing the novel approach for improving the so-called guided filter [12].

Guided filter is categorized into a joint image filtering technique consisting of two input images where one of them is called guidance image, and reflects guidance colors into the other input. One of the most popular joint image filters is the joint bilateral filter [14] which averages the neighbouring colors using the weights that depend on the guidance image. Guided filter is an approach for joint image filtering that allows to overcome a problem with the undesirable gradient reversal artifacts that joint bilateral filter suffers from. Besides edge-aware filtering, it has various image processing applications such as matting, flash-noflash synthesis, HDR-compression, and haze removal.

Consider a point set $\mathbb{X} = \{\mathbf{x}_i\}_{i=1}^N$, $\mathbf{x}_i = (x_i, y_i) \in \mathbb{R}^2$, a guidance image $g = g(\mathbf{x}) \in \mathbb{R}$, an input image $I(\mathbf{x}_i) \in \mathbb{R}$, a desired output image $H(\mathbf{x}_i) \in \mathbb{R}$, and an image region $\Omega(\mathbf{x})$ centered at $\mathbf{x}$. The guided filter is defined as the following linear transformation:

$$H(\mathbf{y}) = ag(\mathbf{y}) + b, \mathbf{y} \in \Omega(\mathbf{x}),$$

where $a, b \in \mathbb{R}$ are the coefficients constant in $\Omega(\mathbf{x})$ that depend on the input image I. Such representation is very useful for image processing tasks, since it preserves the gradient extrema $\nabla H = a\nabla g$, and hence the edges of the guidance image. The coefficients a and b are obtained using the linear ridge regression model [15]:

$$K(a,b) = \sum_{\mathbf{y}\in\Omega(\mathbf{x})} W(\mathbf{x}-\mathbf{y})((ag(\mathbf{y}) + b - I(\mathbf{y}))^2 - \epsilon a^2),$$

where $W(\mathbf{x}-\mathbf{y})$ is the weight that determines the importance of the point $\mathbf{y}$ in $\Omega(\mathbf{x})$ and ϵ is the regularization parameter. One can obtain values a and b by minimizing $K(a,b)$: $\frac{\partial}{\partial a}K(a,b) = 0$ and $\frac{\partial}{\partial b}K(a,b) = 0$. This leads to the following representation:

$$a = \frac{f(Ig) - f(I)f(g)}{f(g^2) - f(g)^2 + \epsilon}, \quad b = f(I) - af(g). \qquad (7)$$

Here $f(*)$ is an averaging function. Since a point $\mathbf{y}$ is included in many overlapping regions $\Omega(\mathbf{x})$ and values a and b for $\mathbf{y}$ are different for each region, the final coefficients are found by averaging over all possible values of $\mathbf{y}$:

$$H(\mathbf{x}) = f(a)g(\mathbf{x}) + f(b). \qquad (8)$$

Guided filtering of color images involves inversion of 3×3 coefficient matrix in order to solve the equation (7) (see [12] for further details). If we set $I \equiv g$, then the guided filter preserves salient edges while smoothing the flat regions (edge-aware filtering). In the simplest case of $I \equiv g$ and I being is a grayscale image, computing guided filter involves performing 4 smoothing operations (e.g. $f(I), f(I^2), f(a), f(b)$), and it takes 33 smoothing operations for a color image if $I \neq g$. Which is why the choice of the smoothing operator $f(*)$ is crucial, since it determines the overall speed and quality of filtering. Authors of the guided filter [12] suggested employing classic L^2 Gauss transform or box kernel method but prefer the latter due to its simplicity and speed despite the fact that box kernel produces undesired artifacts discussed above.

We introduce the new approach for computing guided filtering where our L^1 Gauss transform algorithm is employed for $f(*)$ instead of the box kernel method. As it was shown before, our algorithm gives a much higher quality of smoothing, and this allows us to eliminate smoothing of $f(a)$ and $f(b)$ in the equation (8):

$$H(\mathbf{x}) = ag(\mathbf{x}) + b \qquad (9)$$

Thus, using our algorithm involves 2 operations of $f(*)$ compared to 4 operations in the original method if $I \equiv g$ (grayscale case), and 21 operations compared to 33 operations if $I \neq g$ and both of them are color images.

We examined edge-aware filtering on color images, where the number of $f(*)$ is equal to 21 for the box kernel method and 10 for our approach (9 operations for smoothing of the coefficients and one operation for normalization). As seen on the Figs. 7 and 9, our approach with the reduced amount of smoothing operations $f(*)$ gives quality of edge-aware filtering higher than [12] with the box kernel method, and is faster (0.24 and 0.28 sec for Figs. 9a and 9d respectively).

We examine the differences of equations (8) and (9) in terms of filtering quality on Figs. 9 and 10, which show us that the box kernel method causes artifacts similar to linear filtering case.

We also applied our approach for the detail enhancement filter defined by:

$$D(\mathbf{x}) = I(\mathbf{x}) + \tau(I(\mathbf{x}) - H(\mathbf{x})),$$

where τ is the enhancement parameter. The experiments show that applying our approach for detail enhancement filtering gives high quality results (see Fig. 8).

Fig.10: Edge-aware filtering results (σ=8, ε=0.04). a: input image, b-d: visualization of gradients $|\nabla H|$ of edge-aware filtering via our approach, eq. (9) and box kernel using eqs. (9) and (8) respectively.

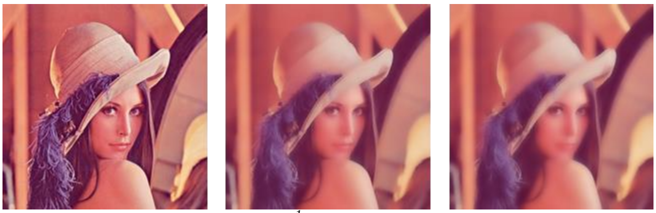
6. Conclusion

In this paper[1] we presented a novel and fast approximation method for L^1 Gauss 2D image transforms. Series of numerical experiments have shown that our method is generally more accurate than the conventional methods and faster than the widely used FFT. We also demonstrated capability of the proposed method in image smoothing application where the conventional box kernel averaging and FFT both suffer from undesirable artifacts. Despite our method is slightly slower than the separable implementations of 1D algorithm [11], this approach can be efficiently used for non-uniformly distributed points.

[1] It is an extension of our previous work [16]. The main difference from [16] is the novel approach to joint image filtering and its numerical experiments.

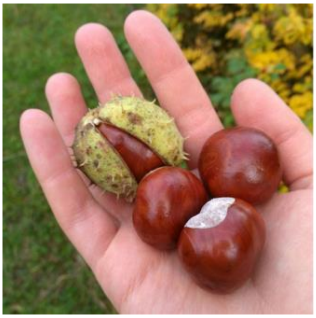
We have also proposed a novel approach for improving the guided filtering [12] via our L^1 Gauss transform and showed its advantages in terms of quality and speed over [12].

Our method is applicable only to uniformly distributed structures, such as images. Hence our future work includes extending the proposed method to higher-dimensional nonuniform cases which can be done for example by using treelike structures. We also would like to investigate possible applications of the proposed method to various machine learning and image processing tasks, such as regression, segmentation, and registration.

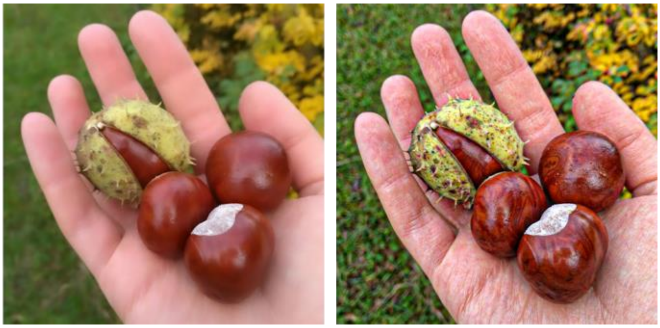


(a) Input *(b) L^1 GT (#f: 10)* *(c) Box (# f: 21)*

Fig. 7: Edge-aware filtering results (σ=8, ε=0.0016).

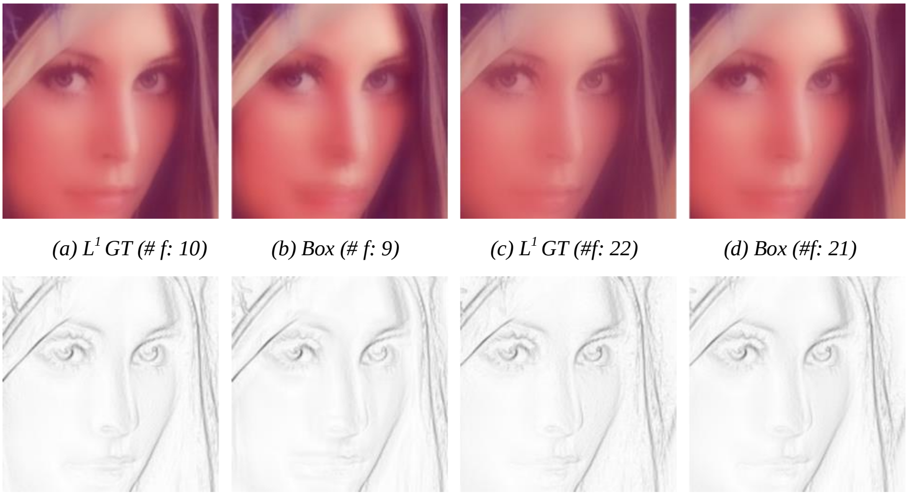


(a) Input



(b) Edge-aware filtering *(c) Detail enhancement*

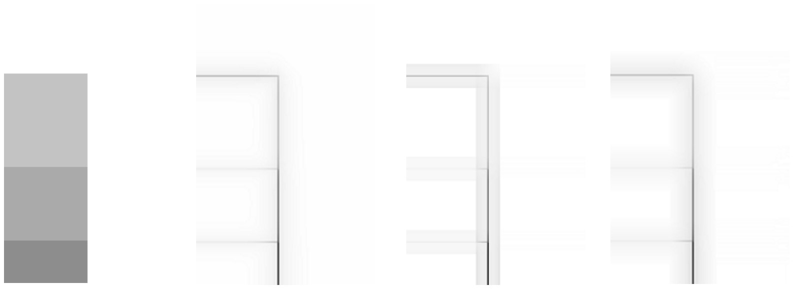
Fig. 8: Our results of edge-aware filtering and detail enhancement (σ=8, ε=0.04, τ=3).



(a) L^1 GT (# f: 10) *(b) Box (# f: 9)* *(c) L^1 GT (#f: 22)* *(d) Box (#f: 21)*

(e) $|\nabla H|$ L^1 GT of (a) *(f) $|\nabla H|$ Box of (b)* *(g) $|\nabla H|$ L^1 GT of (c)* *(h) $|\nabla H|$ Box of (d)*

Fig. 9: Edge-aware filtering results (σ=8, ε=0.0016). a: L^1 Gauss transform with eq. (9), b: using box kernel with eq. (9), c: L^1 Gauss transform with eq. (8), d: box kernel with eq. (8). e-h: visualization of $|\nabla H|$ of the corresponding images.



(a) Input *(b) $|\nabla H|$ Our (#f: 10)* *(c) $|\nabla H|$ Box (#f: 9)* *(d) $|\nabla H|$ Box (#f: 21)*

Acknowledgements

We would like to thank the anonymous reviewers of this paper for their valuable comments. This work was supported in part by Grants-in-Aid for Scientific Research of Japan (15H05954 and 16K15019).

References

[1]. A. Elgammal, R. Duraiswami, and L. Davis, “Efficient kernel density estimation using the fast Gauss transform with applications to color modeling and tracking,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 25, no. 11, pp. 1499–1504, 2003.

[2]. S. Paris and F. Durand, “A fast approximation of the bilateral filter using a signal processing approach,” in *Proc. of European Conference on Computer Vision (ECCV)*. Springer, 2006, pp. 568–580.

[3]. L. Greengard and J. Strain, “The fast Gauss transform,” *SIAM Journal on Scientific and Statistical Computing*, vol. 12, no. 1, pp. 79–94, 1991.

[4]. D. Lee, A. Gray, and A. Moore, “Dual-tree fast Gauss transforms,” *Advances in Neural Information Processing Systems (NIPS)*, vol. 18, pp. 747–754, 2006.

[5]. C. Yang, R. Duraiswami, N. A. Gumerov, and L. Davis, “Improved fast Gauss transform and efficient kernel density estimation.” in *Proc. of International Conference on Computer Vision (ICCV)*, vol. 1, 2003, pp. 464–471.

[6]. A. Adams, N. Gelfand, J. Dolson, and M. Levoy, “Gaussian kd-trees for fast high-dimensional filtering,” in *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3, ACM, 2009.

[7]. E. Dougherty, *Digital Image Processing Methods*. CRC Press, 1994.

[8]. R. Deriche, “Fast algorithms for low-level vision,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 12, no. 1, pp. 78–87, 1990.

[9]. D. Lang, M. Klaas, and N. de Freitas, “Empirical testing of fast kernel density estimation algorithms,” University of British Columbia, Technical Report UBC TR-2005-03, 2005.

[10]. P. Getreuer, “A survey of Gaussian convolution algorithms,” *Image Process. On Line*, vol. 3, pp. 276–300, 2013.

[11]. S. Yoshizawa and H. Yokota, “Fast L^1 Gaussian convolution via domain splitting,” in *Proc. of IEEE International Conference on Image Processing (ICIP)*. IEEE, 2014, pp. 2908–2912.

[12]. He, Kaiming, Jian Sun, and Xiaoou Tang. "Guided image filtering", *IEEE Transactions on Pattern Analysis and Machine Intelligence* (TPAMI), vol. 35, no. 6, pp. 1397-1409, 2013.

[13]. T. Ooura, *General Purpose FFT (Fast Fourier/Cosine/Sine Transform) Package*. www.kurims.kyoto-u.ac.jp/∼ooura/fft.html, 2006.

[14]. Kopf, Johannes, et al. "Joint bilateral upsampling." *ACM Transactions on Graphics (TOG)*, vol. 26. no. 3. ACM, 2007.

[15]. Tikhonov, Andrey. "Solution of incorrectly formulated problems and the regularization method." *Soviet Meth. Dokl.* 1965, Vol. 163, No. 3.

[16]. D. Bashkirova, S. Yoshizawa, R. Latypov and H. Yokota. “Fast L1 Gauss 2D Image Transforms”, *in Proc. of Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE),* Institute for System Programming, RAS, pp. 145-149, 2017. Available at http://syrcose.ispras.ru/2017/SYRCoSE2017_Proceedings.pdf, accessed June 10, 2017.

Быстрое L1-преобразование Гаусса для сглаживания изображений с сохранением границ

[1,2]*Дина Башкирова <dina.bashkirova@riken.jp>*
[1]*Шин Йошидзава <shin@riken.jp>*
[2]*Рустам Латыпов <roustam.latypov@kpfu.ru>*
[1]*Хидео Йокота <hyokota@riken.jp>*
[1]*Image Processing Research Team, RIKEN Center for Advanced Photonics, RIKEN 2-1, Hirosawa, Wako, Saitama, 351-0198, Japan*
[2]*Институт вычислительной математики и информационных технологий, Казанский (Приволжский) Федеральный Университет, 420008 Россия, г. Казань, Кремлевская 35*

Аннотация. Преобразование Гаусса, также как и его дискретный аналог, является важнейшим инструментом во множестве математических дисциплин и находит свое применение во многих научных и инженерных областях, таких как математическая статистика и теория вероятностей, физика, математическое моделирование, машинное обучение и обработка изображений и прочие. Ввиду высокой вычислительной сложности преобразования Гаусса (квадратичная сложность относительно количества точек и экспоненциальная — относительно размерности точек), необходимы эффективные и быстрые методы его аппроксимации, обладающие большей точностью по сравнению с существующими ныне методами, такими как Быстрое Преобразование Фурье или оконное преобразование. В данной статье предложен новый метод аппроксимации преобразования Гаусса для равномерно распределенный множеств точек (например, двумерных изображений), основанный на использовании $\boldsymbol{L^2}$ метрики и метода разделения доменов. Такой подход позволяет значительно сократить количество вычислительных операций путем выполнения предварительных вычислений, и снизить вычислительную сложность метода до линейной. Результаты ряда численных экспериментов показали, что разработанный алгоритм позволяет получить более высокую точность аппроксимации без потери скорости вычисления в сравнении со стандартными методами. Также в качестве примера применения предлагаемого алгоритма была разработана новая схема смежной фильтрации изображения. Было показано, что новый фильтр на основе быстрого $\boldsymbol{L^1}$ преобразования Гаусса позволяет получить результат более высокого качества при сопоставимой скорости вычисления и при этом избежать появления нежелательных артефактов в результате обработки, таких как эффект ореола.

Ключевые слова: фильтр Гаусса, распреледение Лапласа, быстрый метод аппроксимации

DOI: 10.15514/ISPRAS-2017-29(4)-4

Для цитирования: Башкирова Д.Р., Йошидзава Ш., Латыпов Р.Х., Йокота Х. Быстрое L1-преобразование Гаусса для сглаживания изображений с сохранением границ. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 55-72 (на английском). DOI: 10.15514/ISPRAS-2017-29(4)-4

Список литературы

[1]. A. Elgammal, R. Duraiswami, and L. Davis, "Efficient kernel density estimation using the fast Gauss transform with applications to color modeling and tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 25, no. 11, pp. 1499–1504, 2003.

[2]. S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," in *Proc. of European Conference on Computer Vision (ECCV)*. Springer, 2006, pp. 568–580.

[3]. L. Greengard and J. Strain, "The fast Gauss transform," *SIAM Journal on Scientific and Statistical Computing*, vol. 12, no. 1, pp. 79–94, 1991.

[4]. D. Lee, A. Gray, and A. Moore, "Dual-tree fast Gauss transforms," *Advances in Neural Information Processing Systems (NIPS)*, vol. 18, pp. 747–754, 2006.

[5]. C. Yang, R. Duraiswami, N. A. Gumerov, and L. Davis, "Improved fast Gauss transform and efficient kernel density estimation." in *Proc. of International Conference on Computer Vision (ICCV)*, vol. 1, 2003, pp. 464–471.

[6]. A. Adams, N. Gelfand, J. Dolson, and M. Levoy, "Gaussian kd-trees for fast high-dimensional filtering," in *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3, ACM, 2009.

[7]. E. Dougherty, *Digital Image Processing Methods*. CRC Press, 1994.

[8]. R. Deriche, "Fast algorithms for low-level vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 12, no. 1, pp. 78–87, 1990.

[9]. D. Lang, M. Klaas, and N. de Freitas, "Empirical testing of fast kernel density estimation algorithms," University of British Columbia, Technical Report UBC TR-2005-03, 2005.

[10]. P. Getreuer, "A survey of Gaussian convolution algorithms," *Image Process. On Line*, vol. 3, pp. 276–300, 2013.

[11]. S. Yoshizawa and H. Yokota, "Fast L^1 Gaussian convolution via domain splitting," in *Proc. of IEEE International Conference on Image Processing (ICIP)*. IEEE, 2014, pp. 2908–2912.

[12]. He, Kaiming, Jian Sun, and Xiaoou Tang. "Guided image filtering." *IEEE Transactions on Pattern Analysis and Machine Intelligence* (TPAMI), vol. 35, no. 6, pp. 1397-1409, 2013.

[13]. T. Ooura, *General Purpose FFT (Fast Fourier/Cosine/Sine Transform) Package*. www.kurims.kyoto-u.ac.jp/∼ooura/fft.html, 2006.

[14]. Kopf, Johannes, et al. "Joint bilateral upsampling." *ACM Transactions on Graphics (TOG)*, vol. 26. no. 3. ACM, 2007.

[15]. Тихонов, А. Н. "О некорректных задачах линейной алгебры и устойчивом методе их решения." *ДАН СССР, 1965,* 163.3.

[16]. D. Bashkirova, S. Yoshizawa, R. Latypov and H. Yokota. "Fast L1 Gauss 2D Image Transforms", *in Proc. of Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE),* Institute for System Programming, RAS, 2017, pp. 145-149. Доступно по ссылке http://syrcose.ispras.ru/2017/SYRCoSE2017_Proceedings.pdf, дата обращения 10.06.2017.

Real-time digital video stabilization using MEMS-sensors

A.V. Kornilova <kornilova.anastasiia@gmail.com>
I.A. Kirilenko <y.kirilenko@spbu.ru>
N.I. Zabelina <zabelina.nattaly@gmail.com>
Saint Petersburg State University, Software Engineering
28 Universitetskiy prospect, Petergof, Sankt-Peterburg, 198504, Russia

Abstract. This article describes our ongoing research on real-time digital video stabilization using MEMS-sensors. The authors propose to use the described method for stabilizing the video that is transmitted to the mobile robot operator who controls the vehicle remotely, as well as increasing the precision of video-based navigation for subminiature autonomous models. The article describes the general mathematical models needed to implement the video stabilization module based on the MEMS sensors readings. These models includes the camera motion model, frame transformation model and rolling-shutter model. The existing approaches to stabilization using sensors data were analyzed and considered from the point of view of the application in a real-time mode. This article considers the main problems that came up during the experiments that were not resolved in the previous research papers. Such problems include: calibration of the camera and sensors, synchronization of the camera and sensors, increasing the accuracy of determining the camera position from sensors data. The authors offer possible solutions to these problems that would help improve quality of the work of existing algorithms, such as a system for parallel synchronized recording of video and sensor data based on the Android operating system. As the main result, the authors represent a framework for implementing video stabilization algorithms based on MEMS sensors readings.

Keywords: video stabilization; MEMS sensors; real-time system; digital signal processing; computer vision; rolling shutter.

DOI: 10.15514/ISPRAS-2017-29(4)-5

Для цитирования: 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

1. Introduction

Modern cameras' matrices allow to take high-quality pictures that are comparable to professional photographs. However, the quality of video that they are able to record leaves much to be desired and lately it has grown into a problem that needs to be resolved. If modern devices could improve quality of video recording in real time it would not only enable owners of smartphones and action cameras to stream more beautiful and visually appealing video, but would also solve more significant problems. For instance, in case of remotely controlled mobile robots and drones (quadcopters) that perform area monitoring, the low quality of video drastically decreases the precision of control and also leads to greater fatigue of the vehicle operator.

In most cases, you need to get rid of camera shake to solve the problem of poor video quality. It can be achieved either by fixing camera in one place (alternatively, by cancelling out its movement using specially designed mechanisms) or by transforming the frames digitally in such a way so that the video becomes jitterless.

If you choose the first option, you will need special external devices, such as SteadyCam, GyroStick, gimbal (for drones), or specially designed lenses and matrices similar to those available in professional cameras. This approach is not only extremely costly, but also not always applicable. For example, it is impossible to install an external stabilizer on smaller flying vehicles.

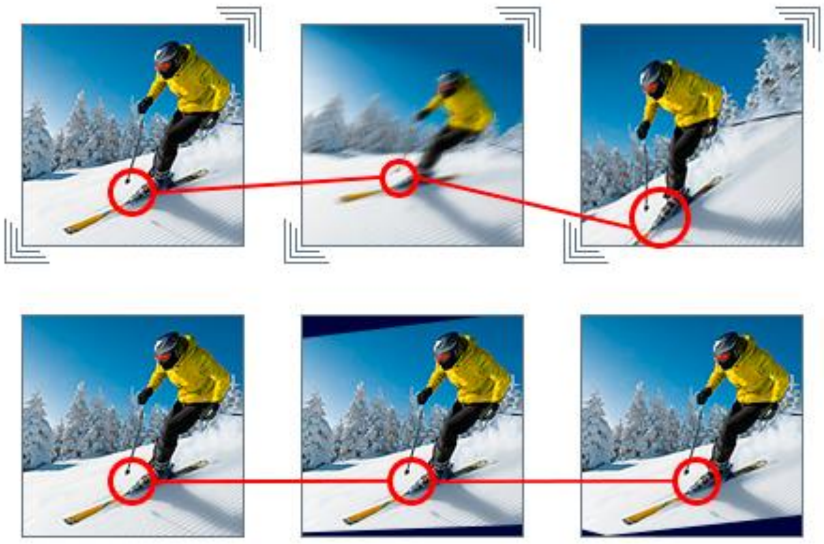


Fig. 1. Image transformation for trajectory smooth

If you opt for the second way, or digital stabilization, you will face the challenge of camera motion estimation and image warping (Fig. 1). Video editing software developers have already advanced significantly in this area. Products like Adobe Premiere, Deshaker, Movavi are all already able to stabilize videos digitally. Similar functionality is also available on YouTube that uses the algorithm proposed in the work [1]. The main disadvantage of these algorithms [2], [3], [4], [5] is the amount of calculations needed to determine the camera motion. This makes this method inapplicable for real-time video stabilization. Besides that, these algorithms only use the data available in the images themselves, which makes them unreliable in case the shot has poor lighting or features large moving objects.

Alternatively, you can estimate the camera motion during the recording by using the information from MEMS (MicroElectroMechanical Systems) motion sensors, including angular rate sensors (gyroscope), accelerometer and magnetometer. This method requires less processing power to determine camera positioning and, consequently, is more energy-efficient, which makes it suitable for real-time video stabilization. For instance, a common gyroscope consumes only 2-5 mW of power. At the same time, the CPU consumes several hundreds of milliwatts while analyzing frames.

This approach is applied more and more in recent years, as MEMS sensors are becoming widespread on different platforms, especially on smartphones. For instance, Google Pixel, introduced in October 2016, completely lacks mechanical stabilization and uses only gyroscope-based stabilization algorithm. IPhone 7 also uses MEMS sensors for video stabilization but employs camera lenses and matrices for this purpose at the same time.

Mobile applications that offer similar functionality are just now coming up on the market and they are only able to perform video stabilization during post-processing. Some of the most prominent ones are: Instagram Hyperlapse, Microsoft Hyperlapse. Gallus is especially noteworthy, because, unlike others, it utilizes data from MEMS sensors.

This article considers different methods of real-time digital video stabilization that utilize MEMS sensors. Given that this research area is located at the junction of computer vision and digital signal processing, a lot of additional tasks arise, that are worth researching both separately and altogether. The main difficulties, when it comes to creating an application that allows to stabilize videos in real time, are the synchronization of frames and sensor data and the creation of a lightweight stabilization algorithm.

Authors review different existing algorithms and approaches as well as describe the problems that surfaced when these methods were implemented. During this research, we have encountered the following challenges: synchronization of frames and sensor reading, efficient frame transformation and increasing the accuracy of camera positioning. This article solves the found problems and offers more stable and universal implementation of the described algorithm.

In the second section of the article, we review the existing approaches to digital video stabilization that utilize MEMS-sensors, analyze whether these algorithms are suitable for use in real time and also list the mathematical models. In the third section, we describe the methods that improve positioning accuracy by using filters and combining readings from different sensors. In the fourth section, we analyze how to efficiently transform frames during camera rotation. In the fifth section, we consider the problem of synchronizing frames and sensor readings and use Android OS as an example. There we also review existing methods of automatic camera and sensor parameters calibration. In the sixth section, we list the main results of the ongoing research.

2. Video stabilization

Video stabilization process can be divided into 3 independent stages:

- estimating camera motion using MEMS sensors;
- calculating the desired camera motion in accordance to some logic (for instance, trajectory smoothing);
- transforming the frame to match camera motion to the desired one.

In order to perform video stabilization in real time, we need to find a solution to each of the above listed tasks that would be satisfactory in terms of quality and performance.

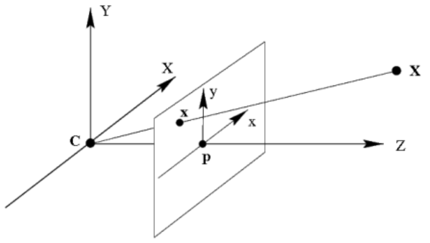
The second stage is the most crucial. When smoothing trajectory, it's important to not only consider jitter as noise, but also to take into account that camera needs to move similarly to the way eye moves naturally. In the beginning of this section, we list the mathematical models and terms that are used and describe the existing algorithms. Then we analyze their advantages and disadvantages, and also propose various improvements.

Authors pay special attention to the two remaining stages, that can be improved significantly, yet still were not touched on in previous papers.

In this section, we suppose that all camera and sensor parameters are known, as well as that sensor readings and camera shots are synchronized in time. The abovementioned problems will be thoroughly discussed in the section dedicated to the parametrization of the stabilization system.

2.1 Mathematical models

Let's take a look at how frame is transformed when camera is rotated. We'll assume that *x* is the coordinates of a point on a projective plane, and *X* is the coordinates of a point in space (Fig. 2). Also, for each particular camera, let's assume that it has the matrix *K* with the following parameters: *(o_x, o_y)* is the optical center of the camera and *f* is its focal length. We'll get the following formulas for the projective transformation [6]:

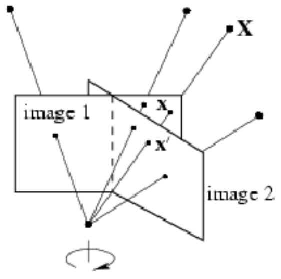


$$x = KX$$

$$K^{-1} = \begin{pmatrix} 1 & 0 & -o_x \\ 0 & 1 & -o_y \\ 0 & 0 & f \end{pmatrix}$$

Fig. 2. Projective transformation

Let's fix the global coordinate system and assume, that in moment *t* the camera is rotated against it, using the rotation matrix *R(t)* (Fig. 3). Then projective transformation will look this way:



$$x = KR(t)X$$

Fig. 3. Location of a point in frames during camera rotation

Let's assume, that x_i и x_j are both projections of the same point *X* in space, but they are located in frames *i* и j respectively, meaning:

$$x_i = KR(t_i)X$$

$$x_j = KR(t_j)X$$

By transforming these expressions, we establish the following connection between projections of the same point in different moments of time:

$$x_j = KR(t_j)R^T(t_i)K^{-1}x_i$$

Thus, let's define the matrix of image transformation between moments in time t_1 и t_2 as:

$$W(t_1, t_2) = KR(t_1)R^T(t_2)K^{-1}$$

$$x_j = W(t_j, t_i)x_i$$

We want to include an additional parameter to the above-described mathematical model of camera and its rotations. It's defined by the camera shutter and solves the problem of blurring when recording fast moving objects. Rolling shutter is a visual distortion that happens, because when the shutter is released, each row of the frame is shot at a different moment in time (Fig. 4-5).

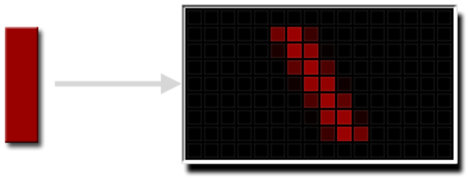


Fig. 4-5. Object movement and Rolling-shutter effect during capturing the moving object

When shutter scans the scene vertically, the moment in time at which each point of frame is shot, is directly dependent on the row it is located in. Thus, if we assume that *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_i 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.

2.2 Stabilization algorithms

Among the solutions discussed in the scientific society, two are especially worth noting, and we will describe them in this section.

2.3 Algorithm with Gaussian filter

Algorithm described in the article [7] in 2011, is based on Gaussian filter. Camera positioning is calculated by integrating the readings of a MEMS gyroscope for each frame. Then the sequence of camera movements is smoothed by utilizing the Gaussian filter (Fig. 6), and the frames are sequenced using the new motion model. Gaussian filter can be customized by changing the window size (how many discrete points it effects) and the size of the core (how strong the smoothing is). By altering these parameters one can either get rid of local jitter or significant movements.

The use of Gaussian filter is very effective during post-processing, but is not always applicable for real-time stabilization. During post-processing movement can be analyzed completely from start to finish, which allows to increase the size of the window of the filter and smooth the movement stronger. During real-time stabilization, processing buffer needs to include 10-15 frames, which results in a significant delay of 0,3-0,5 seconds.

The source code of the prototype was presented in Matlab, but the article states that algorithm was tested on an iPhone 4. During open realization, the algorithm features narrowed camera rotation parameters. Namely, only horizontal camera rotation is taken into account, which does not always reflect the movement of a shaking camera.

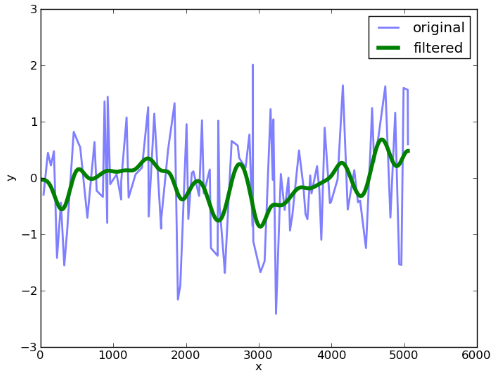


Fig. 6. Trajectory smoothing using the Gaussian filter

2.4 Algorithm utilizing nonlinear filter

Algorithm described in the article [8] in 2014 utilizes a more complex nonlinear filter to smooth camera movement.

In the offered method, the definition of a virtual camera is given. Two concentric zones are selected on the frame – the inner region and the outer region (Fig. 7). Then the rectangle zone is selected in the inner region. Positioning of a virtual camera is determined by the position of this rectangle.

For each new frame, a new position of the abovementioned rectangle is calculated. If it lies within the inner zone, the camera orientation remains the same. If any part of the rectangle lies outside the inner region then the virtual camera’s angular velocity is updated by using spherical linear interpolation to bring it closer the physical camera’s velocity. Authors note that this algorithm works rather well, but when rectangle hits the edge of the inner zone sudden changes can be expected.

The article offers a way how to make this method suitable for real time video stabilization. If a buffer has k frames, than the camera is supposed to move during these frames with the same velocity it did before. If the rectangle crosses the inner zone, then the spherical interpolation is used to bring the virtual camera velocity closer to the velocity of the physical camera.

Besides significantly decreasing the buffer size, this method has one more advantage. It does not take into account the absolute positioning of the camera, as it only uses the velocity of the camera. Therefore, due to the absence of integration, the error is not accumulated.

Sadly, the authors of the article did not offer a repository with source code of the program, realizing this algorithm. Therefore, it was impossible to repeat the experiment at the time. We plan to realize this approach in the nearest future.

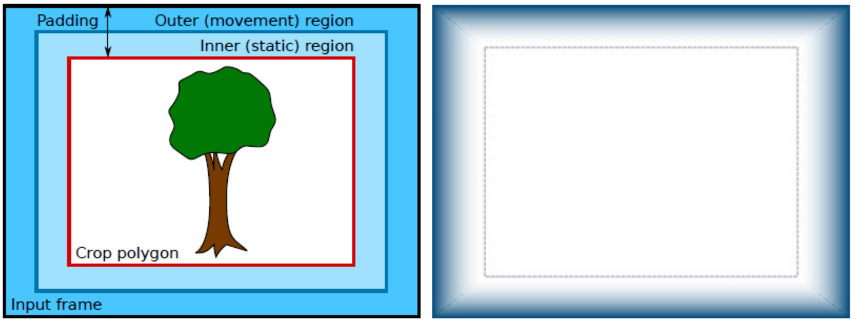


Fig. 7. Inner and outer stabilization zones

3. Determining the positioning

When we were constructing the above-described model, it was assumed that the sensor readings are continuous and accurate. In reality, however, as in all physical devices, MEMS sensors have noise. If the algorithm requires integrating the gyroscope readings, the error caused by the noise will only increase. To solve this problem we will combine the readings of two or more different MEMS sensors, for instance gyroscope and accelerometer. This will allow to eliminate significant errors. The following filters offer similar functionality:

- Complementary filter;
- Madgwick filter [9] – filter that utilizes the gradient descent and allows the use of magnetometer;
- Mahony filter [10];
- Extended Kalman filter – the most successful realization is presented in the work [11].

It is important to mention that the processing complexity of the offered algorithms needs to be minimized for real-time video stabilization. The algorithms are listed in the increasing order of complexity. It is worth noting, that the use of quaternions for estimating positioning and integrating is significantly less complex than other positioning methods like Euler angles or rotation matrices [12].

4. Frame transfomation

After it was determined how much the frame positioning should change, projective transformation should be performed. Realization of the OpenCV library offers this functionality via warpTransoform and perspectiveTransform functions. The first option performs projective transformation for the whole image, while the second one allows to determine the position of particular points on the frame after transformation.

Using the second function allows us to realize the following algorithm. We choose several points on the frame, a 10x10 grid, for instance. After that a projective transformation is performed for each point, and their new positions are calculated (Fig. 8). The values in the other spots are calculated using interpolation.

By varying the size of the grid, it is possible to find the balance between quality of the image after the rotation and speed of processing of the new frame. While experimenting with 1920x1080 frames, it was determined that the best results are achieved with 10x10 grids.

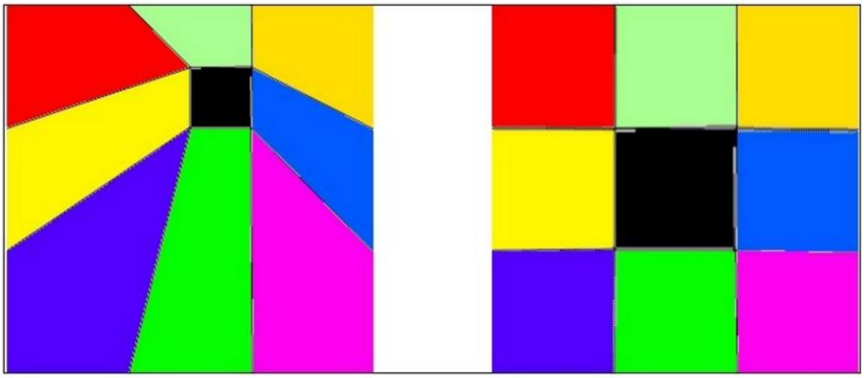


Fig. 8. Image warping

5. Camera calibrations and synchronization

Camera model and the stabilization algorithms, described above, are based on certain assumptions that are not always true in reality. First, it is assumed that sensor readings are a continuous function and are synchronized with frames. Second, we assume that all the necessary parameters for the mathematical model, such as: optical center, focal length and shutter release time are known.

In this section we describe these issues in more detail and offer different solution to the problems.

5.1 Calibrating the unknown camera and sensor parameters

In order for stabilization algorithms to work correctly, we need to have detailed information about camera’s and MEMS sensors’ parameters. Namely, the optical center, focal length, location of the MEMS sensor coordinate axes in relation to the camera’s coordinate axes and shutter release time (rolling shutter). Assuming all pixels are square, we’ll set the optical center at *(0, 0)*.

In case of all sensors, a gyroscope in particular, the main unknown parameter is the *bias* – almost constant skew of angular velocities against the exact measurements. Smartphones sensors are calibrated automatically, while in case of some embedded systems, you need to monitor this parameter closely, as the bias can result in error during integration. To determine the bias, we need to find the mean deviation of angular velocities against the null, when the camera is stable.

The calibration and synchronization problems are solved in the article [13], where the process of online calibration using the extended Kalman filter is described in full detail. Also, in the article [14] the minimization method including determining of the cost function is offered to calibrate the parameters listed above. The full review of camera parameters calibration methods is available in the article [15].

Currently, authors select camera parameters manually for the models used to test algorithms. Automatic calibrations will be realized only after successful experiments with the algorithms.

5.2 Synchronization of a camera and sensors

First, it is important to understand that MEMS sensor readings are discrete. Therefore, even if you know the exact time each frame was taken, it would be impossible to determine the current positioning of the camera. However, since signal’s frequency of the MEMS sensor is between 100 and 200 Hz and the frame rate is 30 fps, we can use simple interpolation to get a relatively accurate estimation.

Unlike embedded systems, that offer hardware synchronization of frames and MEMS sensor reading, operating systems of smartphones sometimes do not offer this functionality. Authors encountered this problem on Android when prototyping the application for simultaneous recording of video and data from sensors.

It turned out, that the main API of the camera, available on each phone does not provide the event scheme for processing single frames. Therefore it was impossible to use software to determine the place of each frame in the time series of sensor readings (Fig. 9). The possible solution to this problem is using the mathematical methods to match two time series with different degrees of discretization: frequent – sensor reading and rare – video frames. The use of displacement of features as metric is suggested.

Starting with level 21 Android API, a new API for Camera2 was introduced. It features the event driven programming that would allow to determine the taking of a frame by using the event handler OnImageAvailableListener. Even if this improvement can’t be used to determine the exact timestamp of a frame, it will help to estimate the place of the frame on the time series of sensor readings. Therefore, this approximation can be used for realizing the mathematical method for matching series.

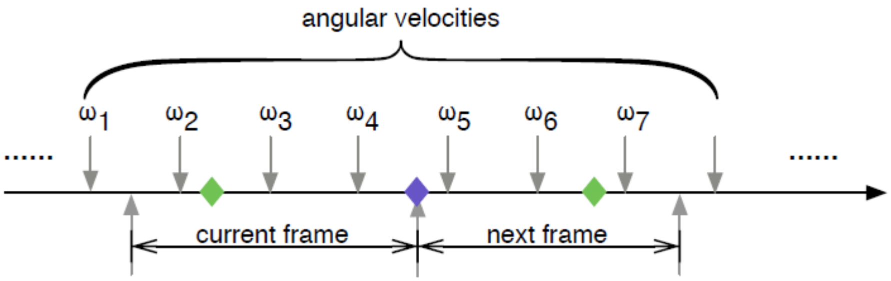


Fig. 9. Matching the time series of frames and gyroscope

6. Current results

Currently, authors have implemented the prototype of the algorithm utilizing the Gaussian filter on Python, that cover the model of 3-dimensional camera rotation. Provided the synchronized sensor readings and frames, as well as intrinsic camera parameters, this algorithm shows great results during post-processing.

Synchronization of sensor readings and camera is performed by an application, described in the corresponding section. Based on this, we plan to execute this algorithm in real-time mode in the nearest future. To decrease latency we will use the optimal filters, that are described in the section dedicated to them, as well as piece-by-piece frame transformation.

To make the software video stabilization module cross-platform, we plan to test the suggested methods of real-time calibration of intrinsic camera parameters and implement them.

7. Conclusion

At this moment, there are many different approaches to digital video stabilization, but not all of them require too much processing power to be used in real time use. Methods utilizing MEMS sensors are worth noting as they allow to save processing resources. Scientific community offers several stabilization algorithms utilizing these sensors. They show great results during post-processing and several prototypes for real-time processing are available.

Despite the possibilities and the need for real-time digital stabilization, its implementation is hard from a technical standpoint, because the video sensor and MEMS sensors need to be coordinated. Besides that, a lot of work still needs to be done to optimize these algorithms for work in real-time.

Many additional challenges and problems described by the authors, show that there is a lot of room for improvement in existing solutions, namely in the way algorithms work. All algorithms that we studied employ a quite primitive mathematical model, which makes it viable to continue research in this area using more advanced mathematics.

Authors set their next goal as using the work they have already done to build a full-fledged software module for real-time digital video stabilization and increase its ability to function on different platforms.

8. Acknowledgment

Funding for this work was provided by JetBrains Research.

References

[1]. Grundmann M., Kwatra V. and Essa I. Auto-directed video stabilization with robust L1 optimal camera paths, CVPR 2011, Providence, RI, 2011, pp. 225-232. DOI: 10.1109/CVPR.2011.5995525.

[2]. Y. Matsushita, E. Ofek, Weina Ge, Xiaoou Tang and Heung-Yeung Shum. Full-frame video stabilization with motion inpainting. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 7, pp. 1150-1163, July 2006.

[3]. Feng Liu, Michael Gleicher, Jue Wang, Hailin Jin and Aseem Agarwala. Subspace Video Stabilization. ACM Transactions on Graphics (presented at SIGGRAPH 2011). Vol. 30, Issue 1, 2011: 4:1-4:10.

[4]. Y. S. Wang, F. Liu, P. S. Hsu and T. Y. Lee. Spatially and Temporally Optimized Video Stabilization. IEEE Transactions on Visualization and Computer Graphics, vol. 19, no. 8, pp. 1354-1361,Aug .2013. DOI: 10.1109/TVCG.2013.11.

[5]. S. Liu, L. Yuan, P. Tan, and J. Sun. Bundled camera paths for video stabilization. ACM Transactions on Graphics (TOG) - SIGGRAPH 2013 Conference Proceedings. Volume 32 Issue 4, July 2013 . Article No. 78. DOI: 10.1145/2461912.2461995.

[6]. R. Szeliski. Computer Vision: Algorithms and Applications. 2010.

[7]. A. Karpenko. Digital Video Stabilization and Rolling Shutter Correction using Gyroscopes. Stanford Tech Report CTSR 2011-03. Stanford University.

[8]. Bell, S., Troccoli, A. J. & Pulli, K. (2014). A Non-Linear Filter for Gyroscope-Based Video Stabilization.. In D. J. Fleet, T. Pajdla, B. Schiele & T. Tuytelaars (eds.), ECCV (4) (p./pp. 294-308), : Springer. ISBN: 978-3-319-10592-5.

[9]. S. Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. Report x-io and University of Bristol (UK), 2010.

[10]. R. Mahony, T. Hamel, J. M. Pflimlin. Complementary filter design on the special orthogonal group SO(3). Proceedings of the 44th IEEE Conference on Decision and Control, 2005, pp. 1477-1484. DOI: 10.1109/CDC.2005.1582367.

[11]. Rong Zhu, Dong Sun, Zhaoying Zhou, Dingqu Wang, A linear fusion algorithm for attitude determination using low cost MEMS-based sensors, Measurement, Volume 40, Issue 3, 2007, Pages 322-328, ISSN 0263-2241.

[12]. J. Diebel. Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors, 2006.

[13]. C. Jia and B. L. Evans. Online Camera-Gyroscope Autocalibration for Cell Phones. IEEE Transactions on Image Processing, vol. 23, no. 12, pp. 5070-5081, Dec. 2014. DOI: 10.1109/TIP.2014.2360120.

[14]. H. Ovrén and P. E. Forssén. Gyroscope-based video stabilisation with auto-calibration. 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, 2015, pp. 2090-2097. DOI: 10.1109/ICRA.2015.7139474.

[15]. Wang Qi, Fu Li and Liu Zhenzhong. Review on camera calibration. 2010 Chinese Control and Decision Conference, Xuzhou, 2010, pp. 3354-3358. DOI: 10.1109/CCDC.2010.5498574.

Стабилизация видеоизображения в режиме реального времени с использованием MEMS-датчиков

А.В. Корнилова <kornilova.anastasiia@gmail.com>
Я.А. Кириленко <y.kirilenko@spbu.ru>
Н.И. Забелина <zabelina.nattaly@gmail.com>
Санкт-Петербургский государственный университет,
Кафедра системного программирования
198504, Россия, г. Санкт-Петербург, г. Петергоф, Университетский проспект, 28

Аннотация. Данная статья описывает текущие исследования по цифровой стабилизации видеоизображения в режиме реального времени с использованием MEMS датчиков. Авторы предполагают использование данного метода для стабилизации видеоизображения, передаваемого оператору дистанционно управляемых мобильных роботов, в частности, для улучшения качества управления малыми летательными аппаратами и снижения усталости оператора. В статье вводятся основные математические модели и понятия необходимые для реализации программного модуля цифровой стабилизации с использованием показаний MEMS датчиков. К таким моделям необходимо отнести: модель вращения камеры, модель трансформации кадра и модель rolling shutter эффекта. Также в статье рассматриваются существующие подходы к стабилизации видеоизображения с использованием MEMS датчиков и дается оценка их применимости в режиме реального времени. Кроме того, освещаются проблемы, возникающие при воспроизведении результатов предыдущих работ и неразрешенные в данных статьях. К таким проблемам следует отнести: синхронизацию показаний датчиков и кадров, калибровку камеры и датчиков, повышение точности определения вращения камеры, эффективную трансформацию кадра при повороте. Авторы предлагают возможные решения данных проблем, в частности, одним из результатов является система параллельной синхронизированной записи кадров и показаний датчиков движения — гироскопа и акселерометра — на базе операционной системы Android. В качестве основного результата представляется фреймворк для тестирования алгоритмов по стабилизации видеоизображения, а также реализация алгоритма стабилизации с использованием фильтра Гаусса для сглаживания траектории движения камеры в рамках данного фреймворка.

Keywords: стабилизация видео; MEMS-датчики; системы реального времени; цифровая обработка сигналов; компьютерное зрение; rolling shutter

DOI: 10.15514/ISPRAS-2017-29(4)-5

Для цитирования: Корнилова А.В., Кириленко Я.А., Забелина Н.И. Стабилизация видеоизображения в режиме реального времени с использованием MEMS-датчиков.

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.

Труды ИСП РАН, том 29, вып. 4, 2017 г., стр. 73-86. DOI: 10.15514/ISPRAS-2017-29(4)-5

Список литературы

[1]. Grundmann M., Kwatra V. and Essa I. Auto-directed video stabilization with robust L1 optimal camera paths, *CVPR 2011*, Providence, RI, 2011, pp. 225-232. DOI: 10.1109/CVPR.2011.5995525.

[2]. Y. Matsushita, E. Ofek, Weina Ge, Xiaoou Tang and Heung-Yeung Shum. Full-frame video stabilization with motion inpainting. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 7, pp. 1150-1163, July 2006.

[3]. Feng Liu, Michael Gleicher, Jue Wang, Hailin Jin and Aseem Agarwala. Subspace Video Stabilization. ACM Transactions on Graphics (presented at SIGGRAPH 2011). Vol. 30, Issue 1, 2011: 4:1-4:10.

[4]. Y. S. Wang, F. Liu, P. S. Hsu and T. Y. Lee. Spatially and Temporally Optimized Video Stabilization. IEEE Transactions on Visualization and Computer Graphics, vol. 19, no. 8, pp. 1354-1361, Aug .2013. DOI: 10.1109/TVCG.2013.11.

[5]. S. Liu, L. Yuan, P. Tan, and J. Sun. Bundled camera paths for video stabilization. ACM Transactions on Graphics (TOG) - SIGGRAPH 2013 Conference Proceedings. Volume 32 Issue 4, July 2013. Article No. 78. DOI: 10.1145/2461912.2461995.

[6]. R. Szeliski. Computer Vision: Algorithms and Applications. 2010.

[7]. A. Karpenko. Digital Video Stabilization and Rolling Shutter Correction using Gyroscopes. Stanford Tech Report CTSR 2011-03. Stanford University.

[8]. Bell, S., Troccoli, A. J. & Pulli, K. (2014). A Non-Linear Filter for Gyroscope-Based Video Stabilization.. In D. J. Fleet, T. Pajdla, B. Schiele & T. Tuytelaars (eds.), *ECCV (4)* (p./pp. 294-308), : Springer. ISBN: 978-3-319-10592-5.

[9]. S. Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. Report x-io and University of Bristol (UK), 2010.

[10]. R. Mahony, T. Hamel, J. M. Pflimlin. Complementary filter design on the special orthogonal group SO(3). Proceedings of the 44th IEEE Conference on Decision and Control, 2005, pp. 1477-1484. DOI: 10.1109/CDC.2005.1582367.

[11]. Rong Zhu, Dong Sun, Zhaoying Zhou, Dingqu Wang, A linear fusion algorithm for attitude determination using low cost MEMS-based sensors, Measurement, Volume 40, Issue 3, 2007, Pages 322-328, ISSN 0263-2241.

[12]. J. Diebel. Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors, 2006.

[13]. C. Jia and B. L. Evans. Online Camera-Gyroscope Autocalibration for Cell Phones. IEEE Transactions on Image Processing, vol. 23, no. 12, pp. 5070-5081, Dec. 2014. DOI: 10.1109/TIP.2014.2360120.

[14]. H. Ovrén and P. E. Forssén. Gyroscope-based video stabilisation with auto-calibration. 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, 2015, pp. 2090-2097. DOI: 10.1109/ICRA.2015.7139474.

[15]. Wang Qi, Fu Li and Liu Zhenzhong. Review on camera calibration. 2010 Chinese Control and Decision Conference, Xuzhou, 2010, pp. 3354-3358. DOI: 10.1109/CCDC.2010.5498574.

Type-2 Fuzzy Rule-Based Model of Urban Metro Positioning Service

A.R. Gimaletdinova <argimaletdinova@edu.hse.ru>
K.Y. Degtiarev <kdegtiarev@hse.ru>
National Research University Higher School of Economics (HSE),
3, Kochnovsky Proezd, Moscow, 125319, Russian Federation

Abstract. In the last few years there has been a growing interest in route building oriented mobile applications with the following features of navigation and sending timely notifications about arrival. Despite the large body of existing knowledge on navigational services, there has been an important issue relative to positioning accuracy. The paper discusses a possible solution to comparison problem, which is linked to the determination of the closeness to destination metro station through finding a difference between user's current coordinates and fixed-point coordinates. With this end in view, fuzzy logic approach is used to develop Routes Recommender System (RRS) that utilizes linguistic variables to express the vague and uncertain term 'closeness to…'. The paper provides detailed explanation of each variable considered in the fuzzy inference system (FIS), set of fuzzy rules in line with graphical representation of system's output. Based on Mamdani model, we propose a set of test cases to check maintainability of the model and provide a description about received results. At a later time, an Android-based mobile application aimed at public transport route building will be developed whose notification system will be based on our model`s implementation presented. It should be emphasized that the paper examines potentials of the modeling approach based on interval type-2 fuzzy sets (IT2FS) that attract much attention these days in various research studies and conventional Mamdani fuzzy inference system (MFIS) as applied to real and rather topical problem. The significance of developing such models may be of a high demand for appropriate representation of factors that are inherently vague and uncertain. Hence, this study may also contribute to future research on similar topics.

Keywords: positioning service; mobile applications; fuzzy modeling; GPS; WiFi; cellular networks; public transport; interval type-2 fuzzy sets; fuzzy inference system; fuzzy matching of coordinates; uncertainty; fuzziness

DOI: 10.15514/ISPRAS-2017-29(4)-6

For citation: Gimaletdinova A.R., Degtiarev K.Y. Type-2 Fuzzy Rule-Based Model of Urban Metro Positioning Service. *Trudy ISP RAN/Proc. ISP RAS,* vol. 29, issue 4, 2017, pp. 87-106. DOI: 10.15514/ISPRAS-2017-29(4)-6

1. Introduction

Over the past decade positioning techniques have become common in almost all branches of industry. In particular, nowadays vast majority of phone models are provided with GPS-module that can be enabled in different cases. Positioning feature is rather common to mobile applications supporting navigational services, and the latter can be used by people in urban transport. The purpose of the paper is to exploit the potentialities of fuzzy logic regarding recommender system with the navigational service. Such service may solve the problem of frequently encountered disorientation of passengers in unfamiliar terrain and allow to pave routes between stations of interest (case of urban transportation system). The potential application may notify a passenger about forthcoming arrival, when he/she is situated closely to the end station. The main purpose in the present context is to determine a deviation between current and end points (stations). Consequently, it leads to the serious problem, since we cannot precisely assert whether a user is close to the end station or not. It occurs because there is a need to estimate the smallest difference (delta) between current and end-point coordinates and then set rule(-s) to classify user`s location.

The issue of applying fuzzy logic to positioning, tracking and transportation attracts attention of researchers. Selected publications have focused on indoor positioning. For example, Chen C.-Y., Yung J., et al. [1] studied indoor positioning technique based on received signal strength and fuzzy approach; they showed experimentally that such method has better performance as compared to geometric triangulation method [2] – actually, the same objective was pursued in the research by Teuber A. and Eisfelller B. [2]. The fuzzy system to control train automatic stop, with the emphasis on stop accuracy, was developed by Yasunobu S., Miyamoto S. and Ihara H. in [3]. It is evident that the practical application of fuzzy logic to positioning or transportation subject matter cannot be considered as exclusive one, however, the issue of positioning in metro should be studied in detail.

As it was mentioned above, the study is devoted to indoor positioning within the metro transportation system. We make an attempt to develop a fuzzy model of metro stops allowing to send timely destination notifications to passenger. It is clear that we do not know exact minimum and maximum distances between stations or the moment when the application should send a reminder. Uncertainty has many faces and forms of manifestation. As stated by George J. Klir and Mark Wierman, “uncertainty involved in any problem-solving situation is a result of some information deficiency; … information may be incomplete, fragmentary, not fully reliable, vague, contradictory, or deficient in some other way” [4]. Hence, when we do not know or cannot obtain exact values/parameters of some phenomena (e.g. distances between points, the location of some moment on a time scale), we need to deviate from type-1 fuzzy sets as a general framework to handle *vagueness* (for more information see seminal papers “Fuzzy Sets” (1965) and “The Concept of a Linguistic Variable and Its Application to Approximate Reasoning – I” (1975) by L. Zadeh) to more general type-2 fuzzy sets that allow to reflect the uncertainty in adequate, more thorough manner, or, put it precisely, to model it. In the work interval type-2 fuzzy sets (IT2FS) are used; to ensure computational efficiency, the preference is given mainly to piecewise linear functions (trapezoidal shape) as upper and lower membership functions of IT2FS.

The rest of the paper is organized as follows: the second section explains the main problem that the paper is devoted to. Section 3 provides definition of linguistic variable (LV) and describes those variables and their linguistic values represented in the form of type-2 membership functions that are used in the inference process (Mamdani's fuzzy model); explanations on domains (universal sets) for each variable are also adduced in this section. The following section 4 makes emphasis on fuzzy rules that serve as a basis for fuzzy system (model developed), covers short comments on type-reduction defuzzification methods used in the study; results of experiments with the system under different values of input variables are presented in both tabular and graphical forms. The last, 5th section of the paper concludes explicitly mentioning the ways of further elaborating upon the subject.

2. Problem definition and general comments

One of the main issues we have to deal with is to find a user's position. Current position obtained should be compared with fixed station's coordinates (e.g. end-point of the route or interchange point to other line) – it will allow to say where is a user now. If he/she is close to one of the points, the application should signal to him about it, thus the understanding and definition of the word "close" becomes essential. The factor of closeness is treated unequally by different people, and a nearby object for one person can be far away for another one. It means that estimation of closeness relates to certain difficulties and, as a consequence, we cannot associate crisp numbers as a basis for possible values of the variable "close". Therefore, the only way to describe closeness at a first approximation is to set a numerical interval of its possible values and to use it at further processing steps.

We may assume that in the beginning the application gets start and end points of the route (input data), then it ensures passenger tracking using one of the positioning

Table 1. Approximate accuracy for different positional techniques [6, 7]

№	Technique	Min accuracy (m)	Medium accuracy (m)	Maximum accuracy (m)
1	GPS	2	11	20
2	WiFi	10	80	150
3	Cellular	100	800	1500
	Average	37.3	297	557
	Average for №2 and №3*	55	440	825

* The last row is calculated without GPS characteristics (signal in metro is bad)

technologies (GPS, WiFi, Cellular Networks) and compares his/her current location with the one of key points. According to [5] and practical everyday experience, GPS has poor accuracy indoors, including metro, therefore, we do not consider GPS positioning accuracy to calculations shown in Table 1. As already mentioned before, we will use numeric interval to represent difference between fixed and current coordinates.

Received data concerning current position can be inaccurate, because positioning techniques used in the phone do not guarantee ′ideal′ precision of geographical coordinates supplied because of various objective reasons (e.g. tracking indoors or underground, inferior quality of signal from provider, etc.). Thus, it makes sense to emphasize another overt source of *fuzziness*, which relates to fuzzy (vague) matching of coordinates – latitude and longitude indicators will be analyzed separately.

3. Fuzzy logic model: definition of linguistic variables and their values

Firstly, we should select input-output variables for fuzzy system and provide necessary explanations. All significant internal and external factors, in which uncertainty shows itself, must be analyzed; this is an important stage in development of the model. Internal factors signify that certain issues depend solely on application itself (its realization), and some tuning steps can lead to better results. On the contrary, external factors indicate that there is obvious reality that is not dependent on realization per se – these are the factors that most people are familiar with, viz. bad quality of signal from provider, poor WiFi coverage, etc.

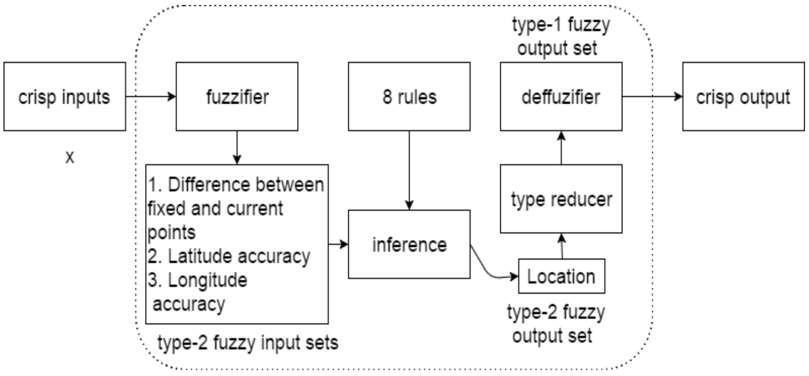


Fig. 1. Fuzzy model with names of linguistic variables in use

In order to explain internal factors, it is necessary to detect variables at the level of passenger’s tracking; the central operation here is obtaining a current position. Once it is done the difference between fixed point coordinates and current point must be determined – we call this difference (i.e. variable) "*Difference between fixed and current points (delta)*".

If we use WiFi and Cellular Networks, it means that the application is going to get coordinates with different accuracy, even at the same place without any movements. We have to admit that the factor of fuzziness definitely becomes apparent in the problem of coordinate matching, and the accuracy will depend on chosen positioning method (see Table 1). We will combine two techniques mentioned above, and because of that Table 1 contains cells with calculated average accuracy. In the paper, we take into account possible accuracy of latitude and longitude – let's name these variables as "*Latitude accuracy*" and "*Longitude accuracy*". The output of fuzzy system will represent position respective to metro station. All these variables as inputs and output of rule-based system to be used are shown in Fig. 1.

Variables mentioned above in the text are *linguistic variables*, i.e. their values are words (or, phrases) of natural language; formally, these values are fuzzy sets, and they are represented by membership functions. In general, a linguistic variable is defined as a tuple $\langle L_{\mathrm{v}}, \mathrm{T}(L_{\mathrm{v}}), \mathrm{U}, \mathrm{G}, \mathrm{M} \rangle$, where L_{v} is the name of the variable (e.g. $L_{\mathrm{v}} \equiv$ "*Latitude accuracy*"), $\mathrm{T}(L_{\mathrm{v}})$ is the set of labels of variable's L_{v} linguistic values $l_1,..,l_{\mathrm{n}}$ (term-set of L_{v}; e.g. $l_{\mathrm{i}} \equiv$ 'insignificant', etc.). The names (labels) are generated using syntactic rule G, the meaning $\mathrm{M}(l_{\mathrm{i}})$ is associated with each value $l_{\mathrm{i}}, \mathrm{i} = \overline{1,\mathrm{n}}$, from $\mathrm{T}(L_{\mathrm{v}})$; $\mathrm{M}(l_{\mathrm{i}})$ is a fuzzy set (respective membership function) defined on a universe of discourse (domain) U. The latter must be defined for all input and output variables introduced earlier. Thus, every variable is characterized by its own set of acceptable values and membership functions for each such value $l_{\mathrm{i}}, \mathrm{i} = \overline{1,\mathrm{n}}$.

3.1 Linguistic variable "delta" and its values

Earlier we were talking about the difference between fixed and current points (so-called *delta*). What does it really mean? The value that expresses the difference falls into the interval $[0, a]$, where real-valued $a > 0$ (deviation is analyzed in absolute magnitude); its left bound (0) means that passenger's coordinates are similar (better to say, close) to some fixed point. We assume that the application should notify a passenger outright before a given destination, when he/she is at the station that precedes terminal station of the route, or at some later moment. Consequently, we consider the average distance between two stations, and a passenger should have enough time to alight from the railway (metro) carriage without effort.

To calculate the biggest difference between coordinates, we should estimate the average distance between any two stations in the metro and double it, because at this moment it will be not an urgent question to notify a passenger about the arrival as he/she still has to go two or more stations more. Following [8, 9], the mean distance between stations in Moscow metro is equal approximately to 1,780 meters. Hence, *a* value signifies the biggest possible difference, i.e. 40,075,000 meters (the length of Earth's equator) = 360° (circle grade measure).

$$1,780 \times 2 = \frac{1,780 \cdot 2 \cdot 360^{\circ}}{40,075,000} \approx 0.032^{\circ} \qquad (1)$$

Therefore, values of *delta* are limited to the interval $[0, 0.032]$ (in degrees) that relates to domain (universal set) U, over which linguistic variable $L_{\text{v}}^{(1)} \equiv "delta"$ is defined. Yet, why do we talk about linguistic variable in that case? In the everyday life people prefer to use words or phrases of the natural language as a habitual terms (values) for description of phenomena they are dealing with in their diverse activities. In case of *delta* variable such attached to it terms as 'big', 'small', etc., on one hand, form a solid ground for communication within the professional medium allowing almost uniform apprehension of the meaning of these values. On the other hand, their inherent uncertainty has to be adequately modeled when used in computational methods. In particular, we may introduce 2 linguistic terms 'small' and 'bigger' (difference between coordinates) as applied to the variable *delta*. Since type-1 membership functions (T1MF) are precise, i.e. the degree of belongingness $\mu(x)$ of each generic element x to corresponding fuzzy set is a crisp number, T1MF cannot represent the typical uncertainty intrinsic to estimates $\mu(x)$ (tilde sign emphasizes the fact that these degrees are not reducible to ordinary numbers).

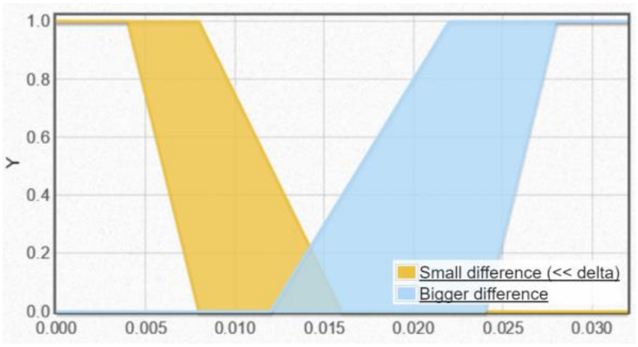


Fig. 2. Difference between fixed and current points (values of "delta")

Linguistic values can be represented in the form of interval type-2 fuzzy sets (IT2FS); the latter are characterized by Lower (L) and Upper (U) membership functions that bound the area called footprint of uncertainty (FOU). The shape of this region allows to express the uncertainty in $\mu(x)$ estimates obtained, providing "additional degrees of freedom … to handle MF uncertainties" [10]. For each $x \in U$, where U is a universe of discourse under consideration, all points in the range $\left[\mu^{(L)}(x), \mu^{(U)}(x)\right]$ may have equal unitary weights, i.e. secondary membership function defined on this interval is constant one. For practical reasons, such IT2FS seem to be convenient enough, accurate from the standpoint of giving proper weigh to uncertainty represented and most easily understood by stakeholders. Henceforth, just this kind of T2FS is used in the model with the direction of attention toward piecewise-linear type (trapezoidal case) of L and U membership functions.

Firstly, it is needed to define trapezoidal MF in terms of L and U functions' parameters for each linguistic value (term) – all calculations are done in accordance with (1). We assume that $L_v^{(1)} \equiv "delta"$ is associated with the term-set $T(L_v^{(1)}) = \{l_1, l_2\} = \{\text{'small difference','bigger difference'}\}$ with 2 elements (Fig.2). The upper function (U) for the term 'small difference' of the variable *delta* can be characterized by parameter's set A(0,0), B(0,1), C(0.008,1) and D(0.016,0); the x-coordinate of the point C is the average of x-coordinates of parameters B (B_x) and D (D_x), the latter is the distance between any 2 stations. In much the same way, for the lower function (L) corresponding parameters are A(0,0), B(0,1), C(0.004,1) and D(0.008,0). The 4-tuple of the upper function (U) that represents the linguistic term 'bigger difference' of *delta* is A(0.012,0), B(0.022,1), C(0.032,1) and D(0.032,0), where $A_x = \left(D_x^{(L\text{'small'})} + D_x^{(U\text{'small'})}\right)/2 = 0.012^\circ$, both x-coordinate C_x and D_x are set to maximum difference 0.032 (1), the value of B_x is calculated as a mean of two neighboring points $\left(A_x + C_x\right)/2 = 0.022^\circ$. It's worth noting that not-yet-application will receive latitude and longitude coordinates as input data, so values are bound to degrees, but not meters. For the lower function (L) set of its parameters takes the form A(0.024,0), B(0.028,1), C(0.032,1), D(0.032,0); again, the value that relates to maximum difference appears here, the B_x value is obtained much as shown above, and A_x equals to the sum of $A_x^{(U\text{'bigger'})}$ and the width of the left tail constituting an approximate half of the distance between stations (890 m) converted to degrees.

3.2 Linguistic variable "latitude/longitude accuracy" and its values (terms)

$L_v^{(2)} \equiv "latitude/longitude\ accuracy"$ is the next variable to consider in the paper. As the telephone receives positioning information due to a correction to be made for the accuracy, it must be taken into account in calculation of difference between fixed and current points. The variable $L_v^{(2)}$ is defined on the interval $[0, b]$, where $b > 0$ is the maximum of average accuracy as shown in the last row of Table 1. The not-yet-application doesn't allow to use GPS in metro, so we consider combined usage of WiFi and Cellular Networks. All calculations shown below are based on values summarized in Table 1, and they are performed in line with (1), i.e.

$$\begin{gathered} 440\text{ m} \approx 0.00395^{\circ};\ 825\text{ m} \approx 0.007^{\circ} \\ 55\text{ m} \approx 0.00049^{\circ};\ \text{mean of min and medium} \\ (55+440\text{ m}) \approx 0.00444^{\circ} \end{gathered} \tag{2}$$

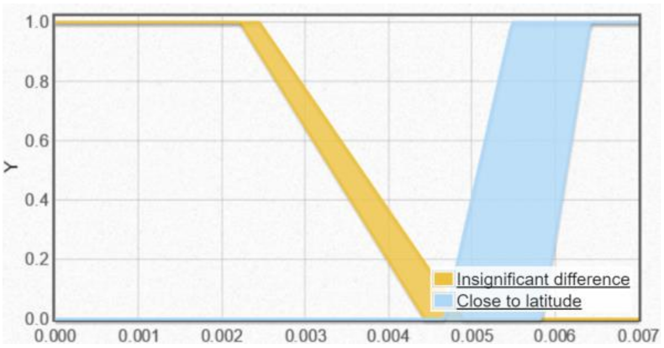


Fig. 3. Values of variables "latitude/longitude accuracy" (same graph)

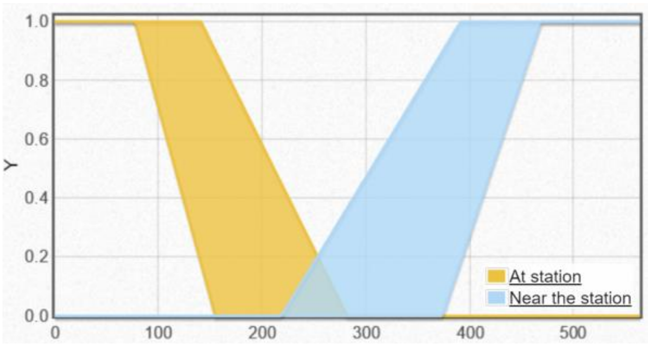


Fig. 4. Two values of the linguistic variable "location"

Thus, the universe, on which variable $L_{\text{v}}^{(2)}$ is defined, results in $\text{U} = [0, 0.007]$ (2).

The upper function (U) for the term 'insignificant difference' of the variable $L_{\text{v}}^{(2)}$ can be characterized by parameter's set A(0,0), B(0,1), C(0.00245,1) and D(0.0049,0); C_x is calculated as the arithmetic mean of B_x and D_x, which is the minimal average accuracy shown in Table 1. As for the lower function (same linguistic term is considered), the values of its parameters are A(0,0), B(0,0), C(0.00222,1) and

D(0.00444,0). First two parameters reflect perfect accuracy at the position; $\mathrm{C_x}$ is obtained as before, while $\mathrm{D_x}$ value corresponds to (2). Linguistic values 'close to latitude/longitude' should be viewed separately, because for each component of coordinate's pair factor of inaccuracy (its measurement) sounds alike, but still differently. Their presence leads to more stable model (Fig. 1) and helps to improve the results attained. The upper function (U) is determined by parameters A(0.00467,0), B(0.00548,1), C(0.007,1) and D(0.007,0), i.e.

$\mathrm{A_x} = \left(\mathrm{D_x^{(L'insig.diff')}} + \mathrm{D_x^{(U'insig.diff')}}\right)\big/2 = 0.00467^\circ$, $\mathrm{B_x}$ is an arithmetic mean of $\mathrm{A_x}$ and $\mathrm{C_x}$. For the lower function (L) parameters are specified as follows: A(0.00584,0), B(0.00642,1), C(0.007,1) and D(0.007,0), where

$\mathrm{A_x} = \left(\mathrm{A_x^{(U'close\ to\ latitude')}} + 0.007^\circ\right)\big/2 = 0.00584^\circ$, $\mathrm{B_x}$ is calculated much as it is done in the case of upper function (U), both $\mathrm{C_x}$ and $\mathrm{D_x}$ are equated with the value of 0.007° that stands for minimum accuracy (or, maximum inaccuracy) – corresponding values are shown in Fig. 3. In the case concerned, only non-negative values of accuracy are considered; if calculations lead to negative result, we use its modulus.

3.3 Linguistic variable “location” and its values (terms)

The variable $L_{\mathrm{v}}^{(3)} \equiv "location"$ is the next matter under discussion – actually, it expresses the location, as it arises from variable's name, of a passenger due to indications related to previously mentioned variables. The variable is represented graphically in Fig. 4. We introduce two values (fuzzy sets) of $L_{\mathrm{v}}^{(3)}$, namely, they are 'at station', i.e. main region that must be reached to notify a user about the arrival, and 'near the station'. The standard length of Moscow metro' platform is appr. 155 meters (8 train carriages), the longest station is “Vorobyovy Gory” – its length is about 282 meters [11]. The universe of discourse U the variable $L_{\mathrm{v}}^{(3)}$ is defined on can be denoted as $[0,c]$, where the right bound *c* equals to the double length of the longest platform in the metro. For the upper function (U) as a constituent of IT2MF representing value 'at station', we set the following parameters: A(0,0), B(0,0), C(141,1) and D(282,0), where $\mathrm{C_x}$ is a half of the longest station (282 m) in the Moscow's metro. The parameters of the lower function (L) of IT2MF are A(0,0), B(0,0), C(77.5,1) and D(155,0) with $\mathrm{C_x}$ calculated as the arithmetic mean of $\mathrm{B_x}$ and $\mathrm{D_x}$ coordinates. We suggest to model the linguistic value 'near the station' with the IT2MF, whose upper function (U) is characterized by A(218.5,0), B(391.25,1), C(564,1) and D(564,0); the value of $\mathrm{A_x}$ is obtained as $\left(\mathrm{D_x^{(L'at\ station')}} + \mathrm{D_x^{(U'at\ station')}}\right)\big/2 = 218.5$ (in meters), $\mathrm{B_x}$ is the mean of $\mathrm{A_x}$ and $\mathrm{C_x}$ x-coordinates, both C_x and D_x are equal to 564 meters (double length of the longest platform). Similarly, parameters of the lower function (L) are A(373.5,0), B(468.75,1), C(564,1) and D(564,0), where A_x (x-coordinate of the first parameter) equals to $A_x^{(\text{U'at station'})} + 155 = 373.5$ that takes into account the length of the standard metro train, i.e. the latter will have direct influence on the spread of the left tail of the membership function. As before, coordinate B_x is the average of A_x and C_x (468.75 meters), and non-negative values are considered.

Rather detailed description of linguistic variables and their values is important for deeper understanding of fuzzy logic system (its model), the use of interval type-2 membership functions to represent uncertainty inherent in verbal values introduced and with the regard for specific character of possible implementation of the system in the code. To a large extent, the definition of a very small number of linguistic variables' values pursues two plain objects – namely, (1) to obtain the initial "non-overloaded" (in terms of number of values and fuzzy rules) variant of the system to perform experiments with and to lay a ground for further analysis, tuning parameters and rule base, revealing drawbacks, etc., and (2) to examine the general idea of using type-2 fuzzy sets in recommendation services that are actively advancing as it applies to enormous market of mobile devices.

4. Rules of the fuzzy model (Inference System) and experiments conducted

The core of the fuzzy inference system (FIS) as shown in Fig. 1 is a set of linguistic values represented in the form of fuzzy sets, If-Then rules having a generic form ″If {*antecedent*} Then {*consequent*}″ and fuzzy reasoning scheme; the latter just operate on a given rules along with specified inputs to derive system's outputs or conclusions. The experts' understanding of the phenomenon under study and their knowledge of the domain field provide a basis for formation of the primary version of rule-base, in which linguistic variables $L_v^{(1)} \equiv \text{"}\textit{latitude/longitude accuracy}\text{"}$ and $L_v^{(2)} \equiv \text{"}\textit{delta}\text{"}$ are used in antecedent part of fuzzy rules (input of the system), whereas $L_v^{(3)} \equiv \text{"}\textit{location}\text{"}$ operates as system's output (its terms form consequent part of rules). The evident transparency of the rule-base in general is substantiated here by a specific fact of simplicity and lucidity of both linguistic values submitted for consideration and existing relations between them. To the opinion of authors, such situation can be viewed as an advantage in terms of efforts needed to design the rule-base. However, it does not mean that the subsequent fine-tuning of rules as well as values of variables will not be needed – most likely, this stage is unavoidable in practice regardless of the system at hand. At the moment, the rules can be represented in the following form:

Rule 1	If *delta* is ′small difference′ and *latitude accuracy* is ′insignificant difference′ and *longitude accuracy* is ′insignificant difference′ Then *location* is ′at station′
Rule 2	If *delta* is ′small difference′ and *latitude accuracy* is ′close to latitude′ and *longitude accuracy* is ′insignificant difference′ Then *location* is ′near the station′
Rule 3	If *delta* is ′small difference′ and *latitude accuracy* is ′insignificant difference′ and *longitude accuracy* is ′close to longitude′ Then *location* is ′near the station′
Rule 4	If *delta* is ′small difference′ and *latitude accuracy* is ′close to latitude′ and *longitude accuracy* is ′close to longitude′ Then *location* is ′near the station′
Rule 5	If *delta* is ′bigger difference′ and *latitude accuracy* is ′insignificant difference′ and *longitude accuracy* is ′insignificant difference′ Then *location* is ′near the station′
Rule 6	If *delta* is ′bigger difference′ and *latitude accuracy* is ′close to latitude′ and *longitude accuracy* is ′insignificant difference′ Then *location* is ′near the station′
Rule 7	If *delta* is ′bigger difference′ and *latitude accuracy* is ′insignificant difference′ and *longitude accuracy* is ′close to longitude′ Then *location* is ′near the station′
Rule 8	If *delta* is ′bigger difference′ and *latitude accuracy* is ′close to latitude′ and *longitude accuracy* is ′close to longitude′ Then *location* is ′near the station′

4.1 Test 1 (difference between fixed and current points (delta))

The first carried out experiment is related to checking the difference between fixed and current points (i.e. linguistic variable ″delta″) under the constant latitude/longitude accuracies equal to 0.00074 (step of delta's change is taken as 0.0032, number of steps equals to 10). IT2MF is an assortment of type-1 membership functions embedded between upper (U) and lower (L) functions. Each of these embedded functions (type-1) can be defuzzified, viz. converted to crisp number that represents generically corresponding fuzzy set (its membership function). The most commonly used method of defuzzification is called centroid [10]. The processing of type-2 fuzzy systems provides for the use of type reduction procedure (TRp) that can be seen as an expanded form of type-1 defuzzification resorting to Extension principle [12]. Each of rules **Rule i**, $i=\overline{1,8}$, "fires" and leads to obtaining output type-2 fuzzy set under a given input data. The union of these output sets and calculation of the centroid of resultant set is the essence of the centroid type reduction. Both theoretical framework and development of type reduction's use in type-2 fuzzy systems were presented in publications by Karnik N.N. and Mendel J.M. [13, 14]. As applied to IT2FS (secondary membership function in that case is constant), TRp becomes simpler in comparison with generalized type-2 sets – the results of experiment (see the data above) using centroid type reduction (**CTR**) defuzzification as summarized in Table 2.

Table 2. CTR defuzzification

№	Difference between fixed and current points (delta)	Location
1	0.0032	84.398
2	0.0064	94.312
3	0.0096	139.576
4	0.0128	282.000
5	0.016	392.995
6	0.0192	392.995
7	0.0224	392.995
8	0.0256	448.347
9	0.0288	460.487
10	0.032	460.487

Table 3. CoSTR defuzzification

№	Difference between fixed and current points (delta)	Location
1	0.0032	84.398
2	0.0064	84.398
3	0.0096	84.398
4	0.0128	275.180
5	0.016	460.487
6	0.0192	460.487
7	0.0224	460.487
8	0.0256	460.487
9	0.0288	460.487
10	0.032	460.487

On the other hand, another TRp called center-of-sets type reducing approach (**CoSTR** – it is a family of defuzzification methods proposed up to now) can be used to substitute the consequent parts of rule-base by singletons at the centroid of corresponding fuzzy sets (Then-part of rules). Subsequent step is connected with obtaining the centroid of type-1 fuzzy set constituted by aforementioned singletons [10]. Calculated values that refer to test data (section IV, item's *A* preamble) are accumulated in Table 3.

It can be noticed that for a particular set of test data centroid TRp demonstrates better (i.e. smoother) approximation of the moderately growing exponential trend. Relative angularity (in Fig.5 it is not so strongly pronounced in comparison with Fig.6 case) relates to the use of piecewise linear (trapezoidal) functions representing fuzzy sets, certain (potential) drawbacks ascribed to rule-base design issues and small number of linguistic terms defined for each variable under consideration. However, even under these circumstances, results of centroid TRp indicate that it is more sensitive to accuracy changes (fine-tuning) than the second TRp. The second graph (Fig.6) visualizes marked broken line consisting of 2 constant levels, and one of those is rather lengthy. To a variable degree, both lines are increasing, and centroid TRp is preferable, since it considers specificity of all functions' values.

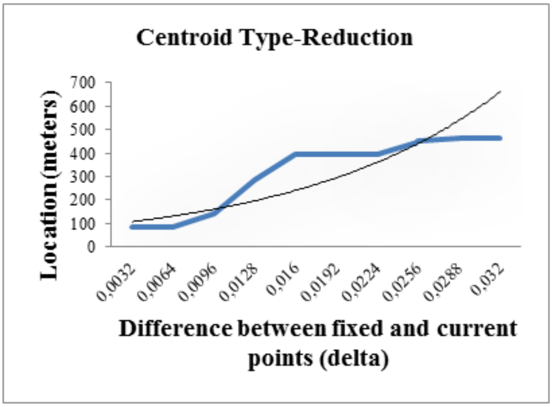


Fig. 5. Centroid type reduction method for "delta" variable

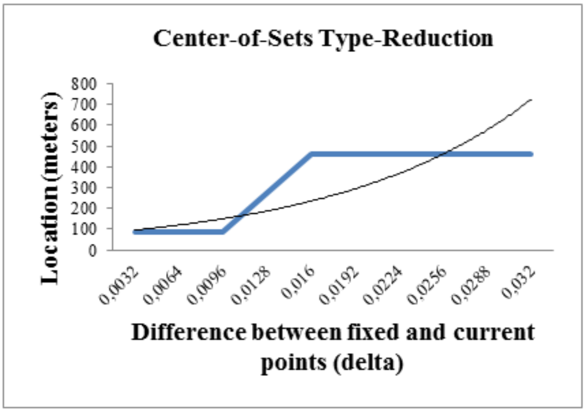


Fig. 6. Center-of-sets type reduction method for "delta" variable

4.2 Test 2 (latitude/longitude accuracy)

The second test relates to checking the latitude/longitude accuracy under constant difference between fixed and current points (delta) equals to 0.0032 (longitude accuracy is 0.00074 OR latitude accuracy is 0.00074, the number of steps is set to 10). Results are shown by Tables 4 and 5.

Here, situation retains characteristic features observed in Fig.5 and 6, i.e. centroid TRp also demonstrates better "behavior". The line (Fig.7) grows monotonously being smooth enough, except for x-coordinates falling into the real range [0.00518, 0.00666] (approx.). Lines shown in both graphs (Fig.7,8) follow the exponential trend (the less latitude/longitude accuracy, the less location accuracy observed).

Table 4. CTR defuzzification

№	Latitude / Longitude accuracy	Location
1	0.00074	84.398
2	0.00148	84.398
3	0.00222	84.398
4	0.00296	90.831
5	0.0037	99.770
6	0.00444	139.576
7	0.00518	392.995
8	0.00592	438.650
9	0.00666	460.487
10	0.0074	460.487

Table 5. CoSTR defuzzification

№	Latitude / Longitude accuracy	Location
1	0.00074	84.398
2	0.00148	84.398
3	0.00222	84.398
4	0.00296	84.398
5	0.0037	84.398
6	0.00444	84.398
7	0.00518	460.487
8	0.00592	460.487
9	0.00666	460.487
10	0.0074	460.487

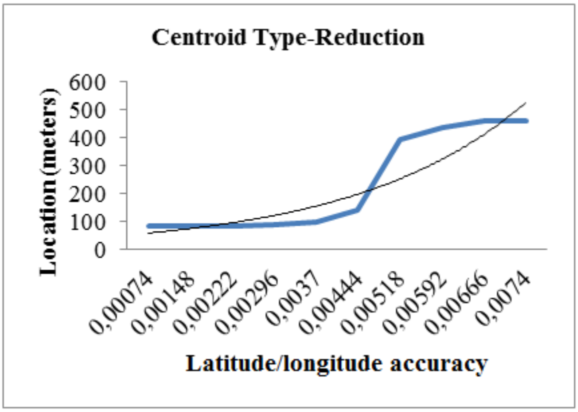


Fig. 7. CTR defuzzification for "latitude/longitude accuracy" variables

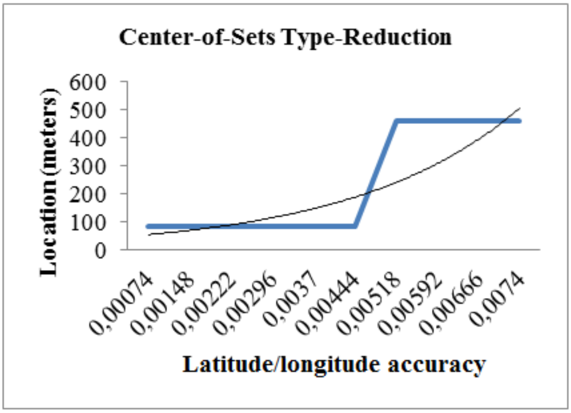


Fig. 8. CoSTR defuzzification for "latitude/longitude accuracy" variables

It should be explicitly mentioned that we have additionally tested rules using two defuzzification methods already mentioned before, namely, (1) centroid TRp and (2) center-of-sets TRp (approaches).

4.3 Test 3 (checking rules used in the model)

Table 6. CTR defuzzification results

№	Rule	Difference between fixed and current points (delta)	Latitude accuracy	Longitude accuracy	Location
1	1*a*	0.0032	0.00074	0.00444	139.576
2	1*b*	0.0064	0.00222	0.0037	99.770
3	1*c*	0.0096	0.00296	0.00296	139.576
4	1*d*	0.0128	0.0037	0.00222	282.0
5	1*e*	0.016	0.00444	0.00074	392.995
6	2*a*	0.0032	0.00518	0.00074	392.995
7	2*b*	0.0064	0.00444	0.00222	139.576
8-22	2*c*-5*b*	0.0096	0.00592	0.00296	392.995
23	5*c*	0.0256	0.00296	0.00296	446.153
24	5*d*	0.0288	0.0037	0.00222	441.641
25-29	5*e*-6*d*	0.032	0.00444	0.00074	392.995
30	6*e*	0.032	0.00666	0.00074	460.487
31-40	7*a*-8*e*	0.0192	0.00074	0.00666	392.995

Each rule was "fed" with 5 (five) test cases, thus each of Tables 6 and 7 covers $40 = 8 \times 5$ cases in total. Tests per rule are numbered in ascending order starting with [n]*a* and ending with [n]*d*, where [n] is the rule's number (index). For example, Rule 3 corresponds to sequence of labelings 3*a*, 3*b*, 3*c*, 3*d* and 3*e* used in Tables 6 and 7. Test data were generated according to intervals of each variable's domain (the same approach as in tests 1 and 2). Step of "*delta*" changes is 0.0032, while step for the latitude and longitude variables equals to 0.00074. Input values are mixed to ensure wider coverage and variety. The last column presents location according to test values and calculation method (TRp) selected. Last column's cells with light-grey shading determine ′At station′ (≤ 282 meters) value (set), while other values show location near some station (linguistic value ′Near the station′). Both tables are wittingly shortened, because of recurrent location results.

Table 7. CoSTR defuzzification results

№	Rule	Difference between fixed and current points (delta)	Latitude accuracy	Longitude accuracy	Location
1	1a	0.0032	0.00074	0.00444	84.398
2	1b	0.0064	0.00222	0.0037	84.398
3	1c	0.0096	0.00296	0.00296	84.398
4	1d	0.0128	0.0037	0.00222	275.180
5	1e	0.016	0.00444	0.00074	460.487
6	2a	0.0032	0.00518	0.00074	460.487
7	2b	0.0064	0.00444	0.00222	84.398
8-40	2c-8d	0.0096	0.00592	0.00296	460.487

A defuzzification method computes the range of possible location values according to input data provided, and the last column of tables shows a mean value of interval bounds, e.g. 139.576 is a mean of [0, 279.152] real-valued range obtained through defuzzification procedure.

5. Conclusion

The paper examined potentials of the modeling approach based on interval type-2 fuzzy sets (IT2FS) and conventional Mamdani fuzzy inference system (MFIS) as applied to real and topical problem related to passengers tracking in urban metro (positioning service by the example of Moscow city). Appeal and significance of developing and further analysis of such models may be of a high demand for appropriate representation of those factors that are inherently vague and uncertain. The aspects that provide for eventuality to discuss models with broad sections of stakeholders owing to model's transparency, abilities to tune their parameters and to carry out experiments (test runs) play a sound role in theory and from practical standpoint. Empirical studies had shown that design issues concerned with linguistic variables and their labelled values (or, terms) influence significantly fuzzy model's output. Test cases presented in the paper corroborate both the applicability and relevance of fuzzy logic-based approach to various problems emerging in the field of navigational services, passenger tracking based on positional technologies. As it was mentioned in section IV, the model that makes use of IT2FS and MFIS leads at the end to resultant intervals that can be calculated in genuine mobile applications without appreciable extra costs with the object of determining the distance to notify users about their arrival (approach) to station. Hence, the developed fuzzy (prototype) model helps to estimate exemplary limits for values of each variable examined. Due to promising test results (for the time being we can talk about model prototype only) and its potential practical applicability, the model (Fig.1) will be implemented in the Android-based mobile program aimed at building routes and notifying users about their destination.

From the standpoint of further theoretical research and topic evolvement, diverse types of membership functions together with fine tuning of their parameters as well as alternative type reduction defuzzification (TRDf) methods should be considered more thoroughly. Besides, by way of illustration GPS technique may beat its own path in IT2FS-based models as applied to ground transportation.

References

[1]. Chen C-Y., Yang J.-P., Tseng G.-J., Wu Y.-H. and Hwang R.-C. An Indoor Positioning Technique Based on Fuzzy Logic, in *Proc. International Multi Conference of Engineers and Computer Scientists (IMECS),* 2010, pp. 854-857.

[2]. Teuber A. and Eissfeller B. WLAN Indoor Positioning Based on Euclidean Distances and Fuzzy Logic, in *Proc. Workshop on Positioning, Navigation and Communication (WPNC),* 2006, pp. 159-168.

[3]. Yasunobu S., Miyamoto S. and Ihara H. A Fuzzy Control for Train Automatic Stop Control. *Transactions of the Society of Instrument and Control Engineers,* 2002, vol. E-2(1), pp. 1-9.

[4]. Klir G.J. and Wierman M. Uncertainty Formalizations. In: Uncertainty-Based Information. Elements of Generalized Information Theory, ser. Studies in Fuzziness and Soft Computing (#15), 2nd ed., Physica Verlag, Germany, 1999, 168 p.

[5]. Arigela L., Veerendra P., Anvesh S. and Hanuman K. Mobile Phone Tracking & Positioning Techniques. *Int. Journal of Innovative Research in Science, Engineering and Technology,* 2012, vol.2, pp. 906-913.

[6]. Gps.gov, "Official U.S. Government Information About the GPS and Related Topics, GPS Accuracy", 2017. [Online resource]. Available: http://www.gps.gov/systems/gps/performance/accuracy/ [Accessed 27-Feb-2017].

[7]. Zeimpekis V., Kourouthanassis P. E. and Giaglis G. M., Mobile and Wireless Positioning Technologies, in *UNESCO Encyclopedia of Life Support Systems (EOLSS),* vol. 6.108, EOLSS Publishers, France, 2007, [http://www.eolss.net].

[8]. Mosmetro.ru, "Metropoliten v tsifrakh", 2017. [Online resource]. Available: http://mosmetro.ru/press/metropoliten-v-tsifrakh/ [Accessed 28-Jan-2017] (in Russian).

[9]. Nashemetro.ru, "Metro v tsifrakh.", 2017. [Online]. Available: http://nashemetro.ru/facts.shtml [Accessed 13-Jan-2017] (in Russian).

[10]. Mendel J.M., Hagras H., Tan W.-W., et al. Introduction to Type-2 Fuzzy Logic Control. Theory and Applications (IEEE Press Series on Computational Intelligence), Wiley-IEEE Press, Piscataway, 2014, 376 p.

[11]. En.wikipedia.org, “Moscow Metro”, 2017. [Online resource]. Available: https://en.wikipedia.org/wiki/Moscow_Metro [Accessed 25-Jan-2017].

[12]. Mendel J.M. Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions, Prentice Hall PTR, Englewood Cliffs, 2001, 576 p.

[13]. Karnik N.N., Mendel J.M. Type-2 Fuzzy Logic Systems: Type-Reduction, in *Proc. IEEE Int. Conference on Systems, Man, and Cybernetics*, 1998, pp. 2046-2051.

[14]. Mendel J.M. Interval Type-2 Fuzzy Logic Systems and Perceptual Computers: Their Similarities and Differences. In: Sadeghian A., Mendel J., Tahayori H. (eds) Advances in Type-2 Fuzzy Sets and Systems. Studies in Fuzziness and Soft Computing, vol. 301. Springer, New York, 2013, pp. 3-18, doi: 10.1007/978-1-4614-6666-6_1.

Модель сервиса позиционирования в метро, основанная на правилах и нечетких множествах второго типа

А.Р. Гималетдинова <argimaletdinova@edu.hse.ru>
К.Ю. Дегтярев <kdegtiarev@hse.ru>
Национальный исследовательский университет
«Высшая школа экономики»,
125319, Россия, Москва, Кочновский проезд, д. 3

Аннотация. За последние несколько лет возник значительный интерес к мобильным приложениям, ориентированным на построение маршрутов пользователей гаджетов; в таких приложениях наряду с важной функцией навигации также возможно отправление своевременных оповещений о прибытии к заданному месту назначения. Несмотря на большой объем имеющейся информации о специфике навигационных сервисов, актуальным остается вопрос относительно точности позиционирования. В данной статье рассматривается возможный подход к решению проблемы сравнения, связанного с определением близости пользователя к конечной станции его маршрута в метро. Такая близость определяется путем подсчета разницы в координатах между текущей позицией пассажира и фиксированной точкой. С целью создания Системы Рекомендаций Маршрутов (СРМ) была применен аппарат нечеткой логики, который использует лингвистические переменные для выражения имеющейся нечеткости (неопределенности) в понимании/восприятии вербального понятия «близость к …». В работе подробно объясняется каждая переменная, используемая в системе нечеткого вывода (англ. FIS), а также представляется набор нечетких правил ЕСЛИ-ТО модели. Для проверки стабильности модели (пока имеет смысл говорить о прототипе модели как первом шаге на пути дальнейшей ее проработки и изменения), основанной на схеме логического вывода Мамдани, рассматриваются несколько тестовых экспериментов с моделью, описываются получаемые результаты. В дальнейшем, планируется разработка мобильного Android-приложения, нацеленного на построение маршрутов городского общественного транспорта с возможностью использования представленной модели при реализации функции по отправлению своевременных оповещений о приближении к пункту назначения. Следует отметить, что акцент делается на использовании в модели интервальных нечетких множеств второго типа (англ. IT2FS), которые привлекают значительное внимание исследователей в настоящее время. Значимость задачи разработки подобных моделей определяется, в первую очередь, необходимостью адекватного учета тех факторов, которые по своей сути являются нечеткими (неопределенными). Данная работа, по мнению авторов, может помочь в продолжении и развитии исследований, связанных с этой же или подобными темами.

Ключевые слова: сервис для позиционирования; мобильные приложения; нечеткое моделирование; GPS; WiFi; мобильные сети; общественный транспорт; интервальные нечеткие множества второго типа; система нечеткого вывода; нечеткость; неточность

DOI: 10.15514/ISPRAS-2017-29(4)-6

Для цитирования: Гималетдинова А.Р., Дегтярев К.Ю. Модель позиционирования в метро, основанная на нечетких правилах второго типа. *Труды ИСП РАН,* том 29, вып. 4, 2017 г., стр. 87-106 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-6

Список литературы

[1]. Chen C-Y., Yang J.-P., Tseng G.-J., Wu Y.-H. and Hwang R.-C. An Indoor Positioning Technique Based on Fuzzy Logic, in *Proc. International Multi Conference of Engineers and Computer Scientists (IMECS)*, 2010, pp. 854-857.

[2]. Teuber A. and Eissfeller B. WLAN Indoor Positioning Based on Euclidean Distances and Fuzzy Logic, in *Proc. Workshop on Positioning, Navigation and Communication (WPNC)*, 2006, pp. 159-168.

[3]. Yasunobu S., Miyamoto S. and Ihara H. A Fuzzy Control for Train Automatic Stop Control. *Transactions of the Society of Instrument and Control Engineers*, 2002, vol. E-2(1), pp. 1-9.

[4]. Klir G.J. and Wierman M. Uncertainty Formalizations. In: Uncertainty-Based Information. Elements of Generalized Information Theory, ser. Studies in Fuzziness and Soft Computing (#15), 2nd ed., Physica Verlag, Germany, 1999, 168 p.

[5]. Arigela L., Veerendra P., Anvesh S. and Hanuman K. Mobile Phone Tracking & Positioning Techniques. *Int. Journal of Innovative Research in Science, Engineering and Technology,* 2012, vol.2, pp. 906-913.

[6]. Gps.gov, “Official U.S. Government Information About the GPS and Related Topics, GPS Accuracy”, 2017. http://www.gps.gov/systems/gps/performance/accuracy/ [Дата обращения 27.02.2017].

[7]. Zeimpekis V., Kourouthanassis P. E. and Giaglis G. M., Mobile and Wireless Positioning Technologies, in *UNESCO Encyclopedia of Life Support Systems (EOLSS)*, vol. 6.108, EOLSS Publishers, France, 2007, [http://www.eolss.net].

[8]. Mosmetro.ru, “Метрополитен в цифрах”, 2017. http://mosmetro.ru/press/metropoliten-v-tsifrakh/ [Дата обращения 28.01.2017] (in Russian).

[9]. Nashemetro.ru, “Метро в цифрах”, 2017. http://nashemetro.ru/facts.shtml [Дата обращения 13.01.2017] (in Russian).

[10]. Mendel J.M., Hagras H., Tan W.-W., et al. Introduction to Type-2 Fuzzy Logic Control. Theory and Applications (IEEE Press Series on Computational Intelligence), Wiley-IEEE Press, Piscataway, 2014, 376 p.

[11]. En.wikipedia.org, “Moscow Metro”, 2017. https://en.wikipedia.org/wiki/Moscow_Metro [Дата обращения 25.01.2017].

[12]. Mendel J.M. Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions, Prentice Hall PTR, Englewood Cliffs, 2001, 576 p.

[13]. Karnik N.N., Mendel J.M. Type-2 Fuzzy Logic Systems: Type-Reduction, in *Proc. IEEE Int. Conference on Systems, Man, and Cybernetics*, 1998, pp. 2046-2051.

[14]. Mendel J.M. Interval Type-2 Fuzzy Logic Systems and Perceptual Computers: Their Similarities and Differences. In: Sadeghian A., Mendel J., Tahayori H. (eds) Advances in Type-2 Fuzzy Sets and Systems. Studies in Fuzziness and Soft Computing, vol. 301. Springer, New York, 2013, pp. 3-18, doi: 10.1007/978-1-4614-6666-6_1.

The Mixed Chinese Postman Problem

M.K. Gordenko <mkgordenko@gmail.ru>
S.M. Avdoshin <savdoshin@edu.hse.ru>
Software Engineering School,
National Research University Higher School of Economics
20, Myasnitskaya, Moscow, 101000, Russia

Abstract. The routing problems are important for logistic and transport sphere. Basically, the routing problems related to determining the optimal set of routes in the multigraph. The Chinese postman problem (CPP) is a special case of the routing problem, which has many potential applications. We propose to solve the MCPP (special NP-hard case of CPP, which defined on mixed multigraph) using the reduction of the original problem into General Travelling Salesman Problem (GTSP). The variants of CPP are pointed out. The mathematical formulations of some problems are presented. The algorithm for reduction the MCPP in multigraph into GTSP is shown. The experimental results of solving MCPP in multigraph through the reduction into GTSP are presented.

Keywords: Mixed Chinese Postman Problem, Arc Routing Problem, heuristic algorithm, Traveling Salesman Problem

DOI: 10.15514/ISPRAS-2017-29(4)-7

For quoting: Gordenko M.K., Avdoshin S.M. The Mixed Chinese Postman Problem. *Trudy ISP RAN/Proc. ISP RAS,* vol. 29, issue 4, 2017, pp. 107-122. DOI: 10.15514/ISPRAS-2017-29(4)-7

1. Introduction

The Chinese Postman Problem (CPP) was originally studied by the Chinese mathematician Kwan Mei-Ko in 1962 on the example of the rural postman problem [1]. A problem is called the CPP after Kwan Mei-Ko [2].

In the modern world, the number of companies and industries that are interested in building an optimal route of product delivery is growing. For example, the postman delivering letters or leaflets wants to know the optimal route that traverses every street in the given area, starting and ending at the office [3].

Apart from the traditional application of the CPP to solving the routing problems such as path planning of snowplows or serving teams, there is a wide range of applications including robot exploration, testing web site usability and finding broken links [3].

There are various classifications of the CPP. This problem can be applied for a directed, undirected, mixed graph, or in a multigraph (a graph with parallel directed and undirected edges). The CPP can also be closed (the postman should return to the starting point) or open (starting and ending points can be different). The problem in directed or undirected graph has exact algorithms and may be solved in polynomial time. The mixed case is NP-hard and there are no polynomial-time algorithms for solving the CPP in mixed graph or multigraph exactly [4, 3].

In this paper, heuristic algorithms for the mixed case are described and assessed. The mixed CPP (MCPP) is a simply-stated problem, which has many useful applications, but has no exact algorithms [3].

The objective of the research is implementation and quality assessment of heuristic algorithms for the MCPP.

The paper is organized as follows. First, the mathematical formulation of the problem is pointed out. The next section is dedicated to related works. In the next part a brief description of implemented algorithms and methodology of the research are presented. Then, already obtained results are revealed. In the final part, the expected results and future directions of research are described.

In this article, in accordance with generally accepted definitions, under the understanding of an understanding directed edge, under the edge is an undirected edge.

2. The General Routing Problem

The routing is one of the most important problem in the optimization researches. The GRP is to define a minimum cost set of routes (one route also is possible) in a multigraph, that must include some required vertices and pass through some required edges and arcs of the original multigraph [1].

Formally, the GRP is defined on multigraph $G =< V, E, A, C >$, where

V is a set of vertices,

A is a multiset of directed edges (arc);

E is a multiset of undirected edges (edges);

C: $E \cup A \to R_+$ is a cost function giving non-negative weights of arcs and edges between vertices.

In the routing problems, it is not necessary to visit all vertices, edges and arcs of the multigraph. Two subsets of edges and arcs $A_R \subseteq A$ and $E_R \subseteq E$ are defined. The arcs and edges from A_R and E_R must necessarily appear in the solution. Let the subset of vertices $V_R \subseteq V$ consist of those vertices that must appear in the route.

The goal of all routing problems is to define a minimum cost set of routes, that traverses all the arcs and edges from the multisets A_R and E_R and includes all vertices of the set V_R.

3. The Vehicle Routing Problem

The VRP is a special case of the GRP with $A_R = \emptyset$ and $E_R = \emptyset$, i.e. the restrictions on the edges and arcs, which must necessarily appear in the route, are absent. The VRP is to determine the Hamiltonian cycle of minimum cost, which traverse all vertices of the subset V_R [1].

In the case, when $V_R = V$, the problem reduces to one of the most famous problem of combinatorial optimization – the classical Traveling Salesman Problem (TSP).

4. The Arc Routing Problem

Another special case of the GRP is the Arc Routing Problem (ARP), it is to determine the minimum cost set of routes, that traverses all required edges E_R and all required arcs A_R of original multigraph. In the ARP, there are no restrictions on the presence of vertices in the route, i.e. $V_R = \emptyset$. The CPP is the variant of ARP. In the original formulation, the CPP is the problem, where the postman should traverse through every street in the given area.

5. The Variants of Chinese Postman Problem

The CPP was originally studied by the Chinese mathematician Kwan Mei-Ko in 1960. A problem is called the Chinese Postman Problem after him. Kwan Mei-Ko defined the problem on undirected graph. Today, there are many various classifications of CPP, including classifications based on the graph type, on the type of solution route and other restrictions and additions [2].

In the modern world, the number of companies and industries that are interested in building an optimal route of product delivery is growing. For example, the postman delivering letters or leaflets wants to know the optimal route that traverses every street in the given area, starting and ending at the office. Apart from the traditional application of the CPP to solving the routing problems such as path planning of snowplows or serving teams, there are a wide range of applications including robot exploration, testing web site usability and finding broken links [3].

Below, the most popular variants of the CPP are presented.

5.1 The Undirected Chinese Postman Problem

The formulation of the Chinese Postman Problem in undirected graph (UCPP) is an original formulation of the CPP problem. The UCPP is a special case of ARP, where
$A = \emptyset$ and $E_R = E$. The UCPP belong to class of P problems.

5.2 The Directed Chinese Postman Problem

The Chinese Postman Problem in directed graph (DCPP) is the modification of UCPP, where every arc (directed edge) can be traversed in one direction. Another name of problem is the New York Street Sweeper Problem. The DCPP is a special case of ARP, where $A_R = A$ and $E = \emptyset$. The DCPP belong to class of P problems.

5.3 The Windy Chinese Postman Problem

The Windy Chinese Postman Problem (WCCP) is the interesting generalization of classical CPP, which has many real uses. In WCPP the cost of traversing some edge depends on way of traversing. The WCPP is a special case of ARP, where $E_R = E$, $A = \emptyset$ and at least for one edge the cost of traversing in direct way is differ from cost of traversing it in opposite way. The DCPP belong to class of NP-hard problems, which cannot be solved in polynomial time.

5.4 The Rural Chinese Postman Problem

The Rural Chinese Postman Problem (RCPP) is a special case of ARP, where $A_R \subseteq A$, $E_R \subseteq E$. Another name of RCPP is the Selecting Chinese Postman Problem. In all above defined CPP problems, it is necessary to find a closed shortest route, that traverses all edges or arcs of the original multigraph at least once. In the real world, it is not always necessary to traverse all roads (edges or arcs). It is enough to traverse only a set of requires arcs and edges (A_R and E_R). The RCPP belong to class of NP-hard problems, which cannot be solved in polynomial time.

5.5 The Mixed Chinese Postman Problem

The Mixed Chinese Postman Problem (MCPP) is a well-known version of the CPP, where multigraph contains both arcs and edges. The MCPP is a special case of ARP, where $E_R = E$ and $A_R = A$. The MCPP belong to class of NP-hard problems.There are other variants of the problem, such as Hierarchical Postman Problem (HCPP), k-Chinese Postman Problem (k-CPP), Chinese Postman Problem with Time Windows and others.

6. The Variants of Chinese Postman Problem

The MCPP is one of the most important problem of the ARP. The MCPP is a special case of ARP, for which $A_R = A \neq \emptyset$, $E_R = E \neq \emptyset$.

An edge $\{v_i, v_j\}$ (an unordered pair of vertices) from the set E is fixed. Let (v_i, v_j) is ordered pair of vertices (this mean, that should traverse $\{v_i, v_j\}$ from vertex v_i to vertex v_j). Note, that, $\forall\{v_i, v_j\} \in E$, $c\big((v_i, v_j)\big) = c\big(\{v_j, v_i\}\big)$ and $C\big((v_j, v_i)\big) = C\big(\{v_j, v_i\}\big)$, it means that the cost of traversing the edge in any directions is the same.

The mathematical formulation of the MCPP problem is presented below.

Let $I = \{1, 2, \dots , |E + A|\}$, $L = \{1, 2, \dots , |V|\}$.

Indexation on the set of vertices V is defined as $inv: V \rightarrow L$, $\forall v_i \in V \ \forall v_j \in V \ v_i \neq v_j \Rightarrow i \neq j$, $i = inv(v_i)$. On the multiset $E \cup A$ indexation is defined as $inea: E \cup A \rightarrow I$, $\forall e_i \in (E \cup A) \ \forall e_j \in (E \cup A) \quad e_i \neq e_j \Rightarrow i \neq j, i = inea(e_i)$.

Route $\mu = (v_{l_1}, e_{p_1}, v_{l_2}, e_{p_2}, \dots, v_{l_n}, e_{p_k})$ is the solution of the MCPP that satisfies to the following properties:

- $e_{p_i} = \left(v_{l_i}, v_{l_{i+1}}\right), i = 1,2, \dots, k-1;$
- $e_{p_k} = (v_{l_k}, v_{l_1});$
- $E \cup A \,/\{e_{p_1}, e_{p_2}, \dots, e_{p_k}\} = \emptyset.$

Let $C(\mu) = \sum_{i=1}^{k} C(e_{p_i})$ is the cost of the MCPP route. Let $\mathcal{M}$ be a set of solutions of the MCPP. It is necessary to find a route $\mu_0 \in \mathcal{M}$ that satisfies the following property $\forall \mu \in \mathcal{M} \ C(\mu_0) \leq \mathrm{C}(\mu)$ or $C(\mu_0) = \min_{\mu \in \mathcal{M}}(\,\mathrm{C}(\mu))$.

7. The reduction of Chinese Postman Problem

The MCPP can be reduced to an equivalent ARP. When problem defined in directed graph (DCPP), it can be reduced to the asymmetric TSP. When problem defined in mixed or undirected graph, it can be reduced to GTSP [4-6].

7.1 Description of reduction algorithm

Originally, the reduction algorithm was presented for the graph [3, 4]. The algorithm modification, applicable to the multigraph, is given below [7].

The process of reduction the MCPP to GTSP is to transform the original graph $G =< V, E \cup A, C >$ into equivalent GTSP on complete graph $\tilde{G} =< \tilde{V}, \tilde{A}, \tilde{C} >$.

Table 1. Formulas for computing arc costs of Asymmetric GTSP

		1	2	3	4	5	6	7	8
		v_{12}^1	v_{13}^1	v_{13}^2	v_{21}^1	v_{23}^1	v_{23}^2	v_{31}^1	v_{32}^1
1	v_{12}^1	-	$s_{21} + c_{13}^1$	$s_{21} + c_{13}^2$	$s_{22} + c_{21}^1$	$s_{22} + c_{23}^1$	$s_{22} + c_{23}^2$	$s_{23} + c_{31}^1$	$s_{23} + c_{32}^1$
2	v_{13}^1	$s_{31} + c_{12}^1$	-	$s_{31} + c_{13}^2$	$s_{32} + c_{21}^1$	$s_{32} + c_{23}^1$	$s_{32} + c_{23}^2$	$s_{33} + c_{31}^1$	$s_{33} + c_{32}^1$
3	v_{13}^2	$s_{31} + c_{12}^1$	$s_{31} + c_{13}^1$	-	$s_{32} + c_{21}^1$	$s_{32} + c_{23}^1$	$s_{32} + c_{23}^2$	$s_{33} + c_{31}^1$	$s_{33} + c_{32}^1$
4	v_{21}^1	$s_{11} + c_{12}^1$	$s_{11} + c_{13}^1$	$s_{11} + c_{13}^2$	-	$s_{12} + c_{23}^1$	$s_{12} + c_{23}^2$	$s_{13} + c_{31}^1$	$s_{13} + c_{32}^1$
5	v_{23}^1	$s_{31} + c_{12}^1$	$s_{31} + c_{13}^1$	$s_{31} + c_{13}^2$	$s_{32} + c_{21}^1$	-	$s_{32} + c_{23}^2$	$s_{33} + c_{31}^1$	$s_{33} + c_{32}^1$
6	v_{23}^2	$s_{31} + c_{12}^1$	$s_{31} + c_{13}^1$	$s_{31} + c_{13}^2$	$s_{32} + c_{21}^1$	$s_{32} + c_{23}^1$	-	$s_{33} + c_{31}^1$	$s_{33} + c_{32}^1$
7	v_{31}^1	$s_{11} + c_{12}^1$	$s_{11} + c_{13}^1$	$s_{11} + c_{13}^2$	$s_{12} + c_{21}^1$	$s_{12} + c_{23}^1$	$s_{12} + c_{23}^2$	-	$s_{13} + c_{32}^1$
8	v_{31}^2	$s_{21} + c_{12}^1$	$s_{21} + c_{13}^1$	$s_{21} + c_{13}^2$	$s_{22} + c_{21}^1$	$s_{22} + c_{23}^1$	$s_{22} + c_{23}^2$	$s_{23} + c_{31}^1$	-

Each arc $a_{ij}^k \in A$ between to vertices $v_i \in V$, $v_j \in V$ is represented as vertex $v_{ij}^k \in \tilde{V}$, which must be used in the solution at least once, where k is the serial number of parallel arc. Each edge $e_{ij}^k \in \tilde{E}$ between to $v_i \in V$, $v_j \in V$ is represented as two vertices $v_{ij}^{k_1} \in \tilde{V}, v_{ij}^{k_2} \in \tilde{V}$, one of which must be used in the solution, another may not be used, where k is a serial number of parallel edge, k_1, k_2 are the serial numbers of parallel arcs between two vertices. After replacing the arcs and edges in vertices, the cost from each pair of vertex $v_{ab}^{k_1} \in \tilde{V}, v_{cd}^{k_2} \in \tilde{V}$ in graph $\tilde{G}$ compute, as $\widetilde{c_{ij}} = d_{bc} + c_{cd}^{k_2}$, where d_{bc} is the shortest distance between vertices $v_b \in V$, $v_c \in V$ in original multigraph G. Then, the compete graph $\tilde{G}$ is partitioned into clusters as follows: each arc and each edge is separate cluster. The number of clusters is equal to $|A \cup E|$. The graph partitioned into clusters because edge can be traverse in two ways, for solving the MCPP any way is appropriate and the problem transforms into GTSP [5-7].

The GTSP is a variation of the Traveling Salesman Problem in which all vertices are divided into clusters, and solution consist from only one vertices from each cluster.

7.2 The example of reduction

The example of original multigraph is shown on Fig. 1. Each arc and edge has the cost of traverse. Each vertex has the serial number.

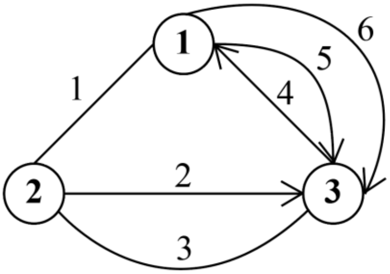


Fig. 1. The original MCPP problem in multigraph

We replace each edge by a pair of two oppositely directed arcs and specify the numbering of parallel arcs between each pair of vertices (see Fig. 2). In multigraph only one arc a_{12}^1 or a_{21}^1 is required, because these arcs represent one edge. The same applies to arc a_{23}^2 or a_{32}^1.

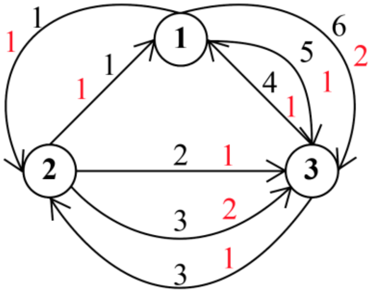


Fig. 2. The results of numbering each parallel arc

After that, should replace each arc and edge as vertex. We received new graph $\tilde{G}$ with 8 vertices. The $\tilde{V}$ can be calculates according to the formula $|\tilde{V}| = |\tilde{A}| + 2|\tilde{E}|$.

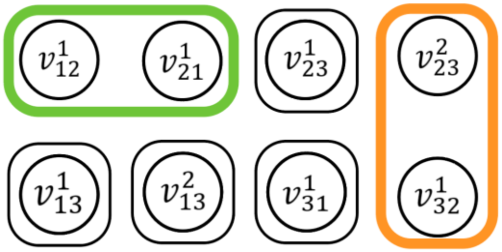


Fig. 3. The vertices and clusters of transformed problems

The cost from each pair of vertices is calculated by formulas (see Table 1, see Table 2). The vertices represent the edge are marked with a color in the table (different colors for different edges).

Table 2. The cost matrix

		1	2	3	4	5	6	7	8
		v_{12}^1	v_{13}^1	v_{13}^2	v_{21}^1	v_{23}^1	v_{23}^2	v_{31}^1	v_{32}^1
1	v_{12}^1	-	6	7	1	2	3	6	5
2	v_{13}^1	5	-	10	4	5	6	4	3
3	v_{13}^2	5	9	-	4	5	6	4	3
4	v_{21}^1	1	5	6	-	3	4	7	6
5	v_{23}^1	5	9	10	4	-	6	4	3
6	v_{23}^2	5	9	10	4	5	-	4	3
7	v_{31}^1	1	5	6	2	3	4	-	6
8	v_{32}^1	2	6	7	1	2	3	6	-

Then vertices from $\tilde{V}$ are partitioned into clusters. Fig. 3 depicts the vertices and clusters of transformed graph. The reduction of the MCPP to Asymmetric GTSP is received. After transformation of the original MCPP into the GTSP, the existing algorithm for GTSP can be applied.

8. Algorithms of GTSP Solving

In the work, the nearest neighbor heuristic algorithm (NN) and its modifications were applied to solve the GTSP problem.

8.1 Nearest Neighbor Heuristic (NN)

Nearest Neighbor heuristic belongs to the group of tour construction heuristics. In the tour construction methods, the route is built by adding new vertices at each step, according to some rules, while the already existing tour does not improve.

The algorithm starts building the route from some starting vertex, and then selects the nearest vertex from another cluster to the start point and adds it to the route. Then, the nearest vertex, belonging to an unused cluster, should appear in the route, until all the clusters are used. After adding the vertices, we return to the starting point.

Time complexity of the algorithm in the best and worst case – $O(|\tilde{\tilde{V}}|^2)$ [5].

8.2 Repetitive Nearest Neighbor Heuristic (RNN)

Since the length of the obtained route in NN depends on the considered starting vertex, another variant of the nearest-neighbor is the repeated nearest neighbor, which calculates the cost of the route, when NN is applied to each vertex as starting vertex, and chooses the best route among all.

Time complexity of the algorithm in the best and worst case – $O(|\tilde{\tilde{V}}|^3)$ [5].

8.3 Improved Nearest Neighbor Heuristic (INN)

Another modification of NN is the heuristic, in which the shortest edge between two vertices from different clusters is selected as the starting edge for the route, and then NN is applied to the found edge (the end vertex of shortest edge is the starting vertex for NN).

Time complexity of the algorithm in the best and worst case – $O(|\tilde{\tilde{V}}|^2)$ [5].

8.4 Repetitive Improved Nearest Neighbor Heuristic (RINN)

This method is a joint modification of the methods RNN and INN, which based on the fact, that in the problem there are several edges with a minimum weight. It is proposed to find the lengths of routes for all minimal edges [6].

Time complexity of the algorithm in the best case is $O(|\tilde{\tilde{V}}|^2)$ and in the worst case is $O(|\tilde{\tilde{V}}|^3)$ [6].

8.5 Double-Ended Nearest Neighbor Heuristic (DENN)

The algorithm starts building the route from some starting vertex, and, then, selects the nearest vertex to the start vertex and adds it to the route. Then the nearest vertex, belonging to an unused cluster, to the first vertex in the solution or the last is added, should appear in the route, until all the clusters are used. Thus, the route grows from both ends, the vertices can be added at the beginning of the route, and at the ending of the route. After adding all the vertices, we return to the starting point.

Time complexity of the algorithm in the best and worst case – $O(|\tilde{\tilde{V}}|^2)$ [5].

8.6 Loneliest Nearest Neighbor Heuristic (LNN)

The main idea of the heuristic is that the vertices most remote from the others should be paid special attention during the construction of the route to avoid their later inclusion in the route with higher cost. To make such heuristic possible, the concept of “loneliness” of the city was introduced. Together with the distance to the nearest neighbor, the closeness of the nearest neighbors will also be the criteria for adding the next vertex to the route. “Lonely” neighbors, i.e. most remote, will be preferable to others. At the preprocessing stage, a new distance matrix is obtained, such that shorter new distances from the vertex to the others are a weighted function of short old distances to these vertices, and a higher loneliness of this city. Then NN is applied to the new matrix [5].

Time complexity of the algorithm in the best and worst case – $O(|\tilde{\tilde{V}}|^2)$.

8.7 Double-Ended Nearest Neighbor Heuristic (DENLN)

The heuristic is a modification of the NLN and DENLN heuristics, the weighted distance function is also calculated here, considering the “loneliness” of the city, but the DENLN algorithm is already applied in the next stages [5].

Time complexity of the algorithm in the best and worst case – $O(|\tilde{\tilde{V}}|^2)$ [6].

9. Methods of testing

The algorithm for graph transformations and solving GTSP was written on C++ in MS Visual Studio 2015.

To test all algorithms, the two databases were used. For multigraph, the test data sets were not found. However, graph is a special case of multigraph (without parallel arcs and edges) and algorithm can be tested on graph data sets. In Bönisch’s database the input data for 50, 100, 200 vertices in graph are presented [8]. For each dimension, there are 75 different tasks. The test data from Angel Corberan web-site for 500, 1000, 1500, 2000 and 3000 vertices also was used [9]. For each dimension, there are 25 different tasks.

Table 3. The Pareto-optimal algorithms for solving MCPP through the reduction to GTSP

	$\|V\| = 50$	$\|V\| = 100$	$\|V\| = 200$	$\|V\| = 500$	$\|V\| = 1000$	$\|V\| = 1500$	$\|V\| = 2000$	$\|V\| = 3000$
NN			+		+	+		
INN	+	+	+	+	+	+	+	+
RINN	+							+
DENN								
NLN						+		
DENLN								
RNN	+	+	+	+	+	+	+	+

The time performance and error rate of proposed approach were measured as follows:

- Test data were loaded in console program.
- The measurements for each input data set were carried out 10 times. The results of computational time were obtained as the average of 10 runs of the program: $T_{av} = \frac{T_1+\cdots+T_{10}}{10}$.
- Error rate of the TSP algorithms was evaluated according to the formula $Error = \frac{C(\mu)-C(\mu_0)}{C(\mu_0)}$, where $C(\mu)$ is the resulting length of the route of the MCPP, $C(\mu_0)$ is the optimum length of the route of the MCPP given in input data.

All test provides on Mac Book Pro 13 retina 2014 (Intel Core i5, 2.6 GHz).

10. Obtained results

For all tests from test database the time and error rate were computed. On Fig. 4, Fig. 5, Fig. 6, Fig. 7, Fig. 8, Fig. 9, Fig. 10, Fig. 11 the results are presented in the form of diagrams, where for each described above algorithm the average computational time and error rate are depicted.

Diagrams allow determine the Pareto-Optimal algorithms on two criteria: computational time and error rate. For all tested dimension, this groups are similar and contain RNN and INN algorithms.

The obtained results make it possible to conclude that the proposed approach is applicable to MCPP and gives good results in terms of computational time and error, taking into account the fact, that one of the simplest heuristics was used (the nearest neighbor heuristic).

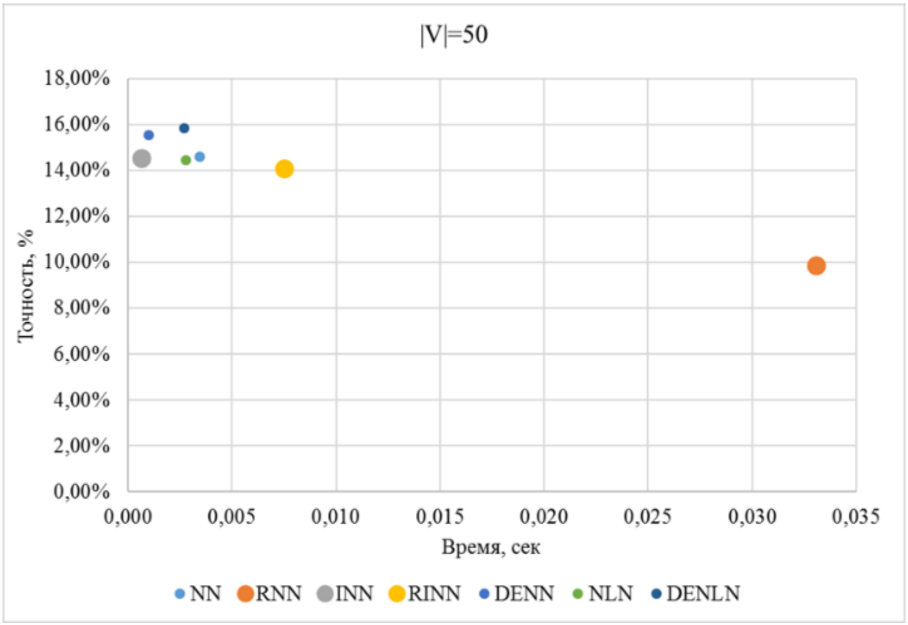


Fig. 4. The results for |V|=50

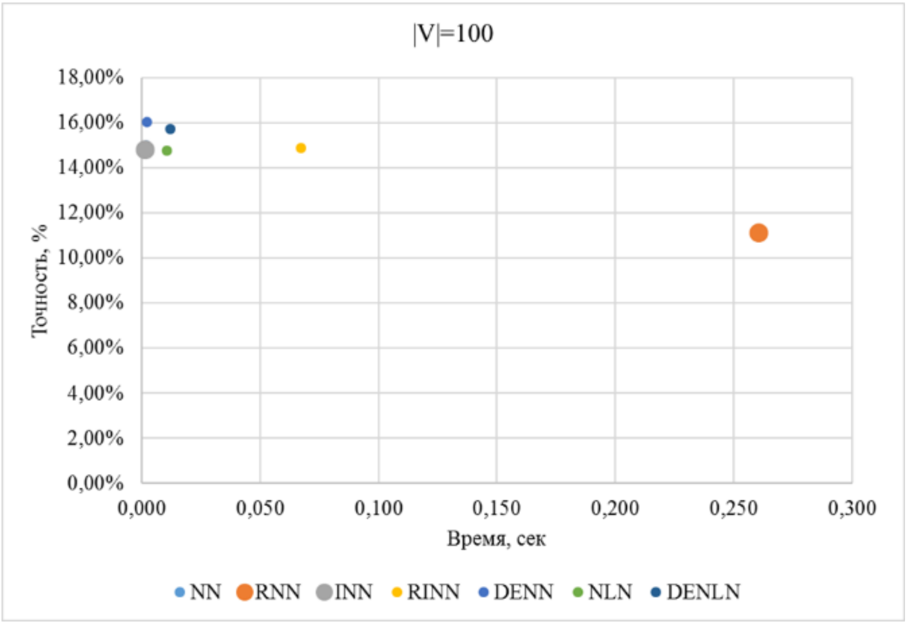


Fig. 5. The results for |V|=100

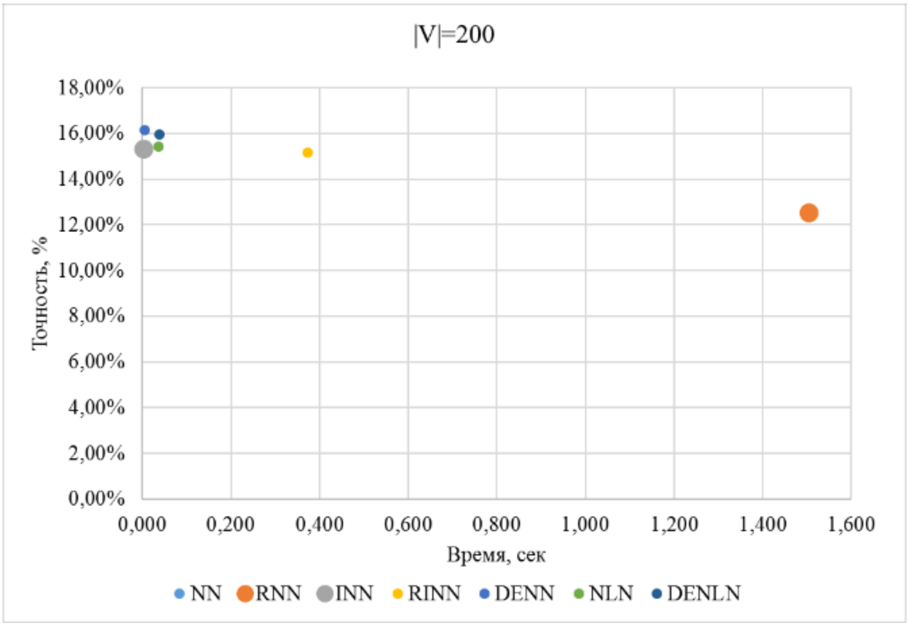


Fig. 6. The results for |V|=200

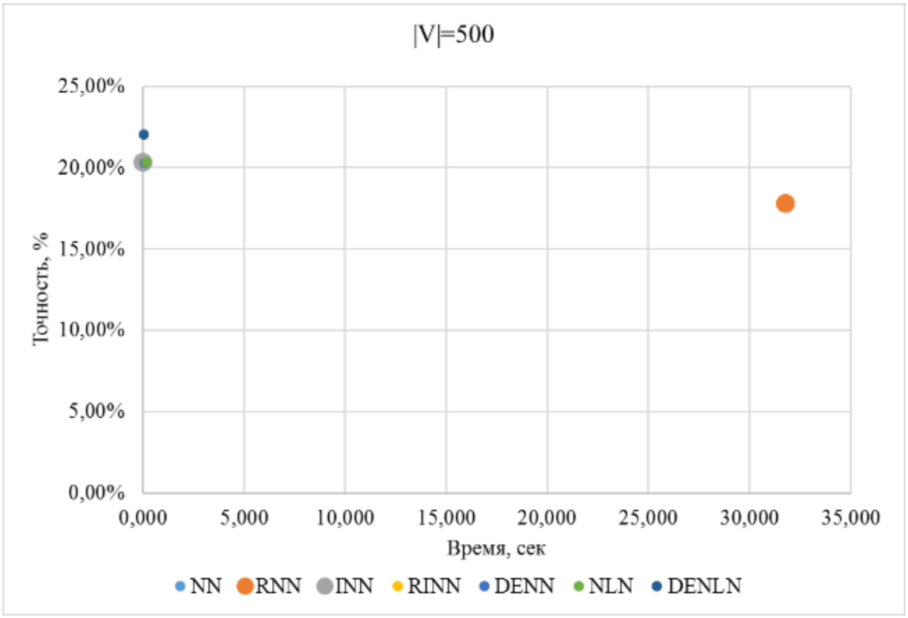


Fig. 7. The results for |V|=500

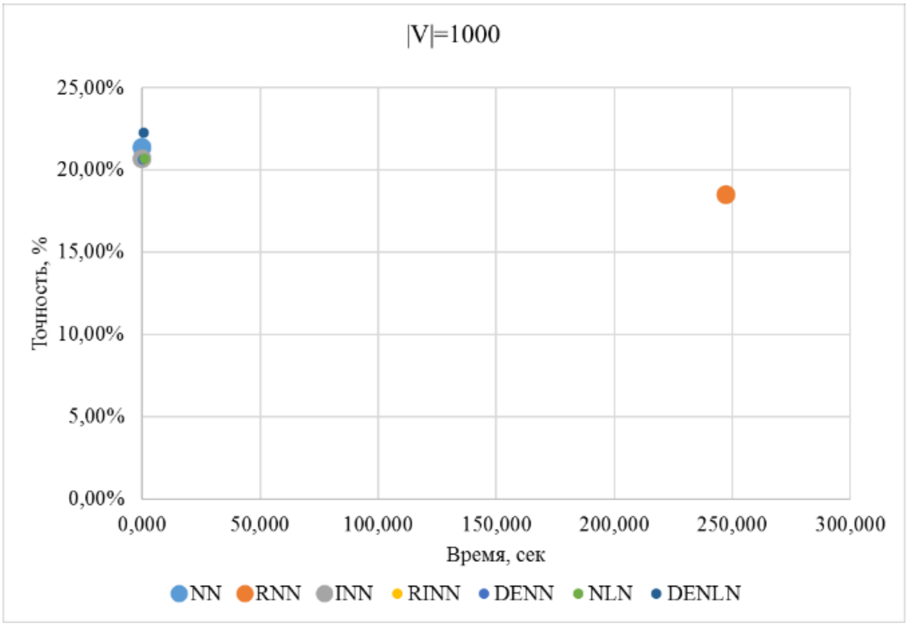


Fig. 8. The results for |V|=1000

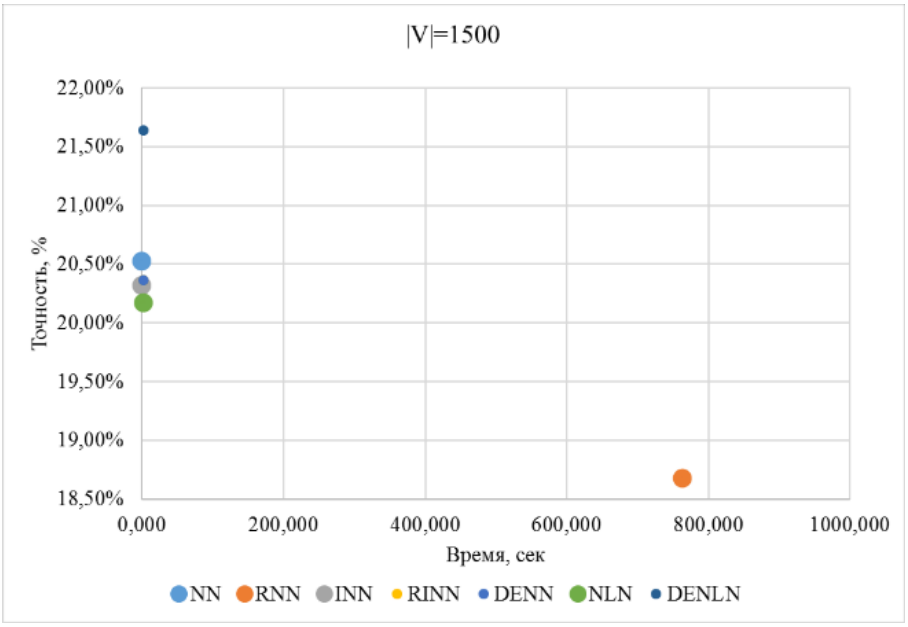


Fig. 9. The results for |V|=1500

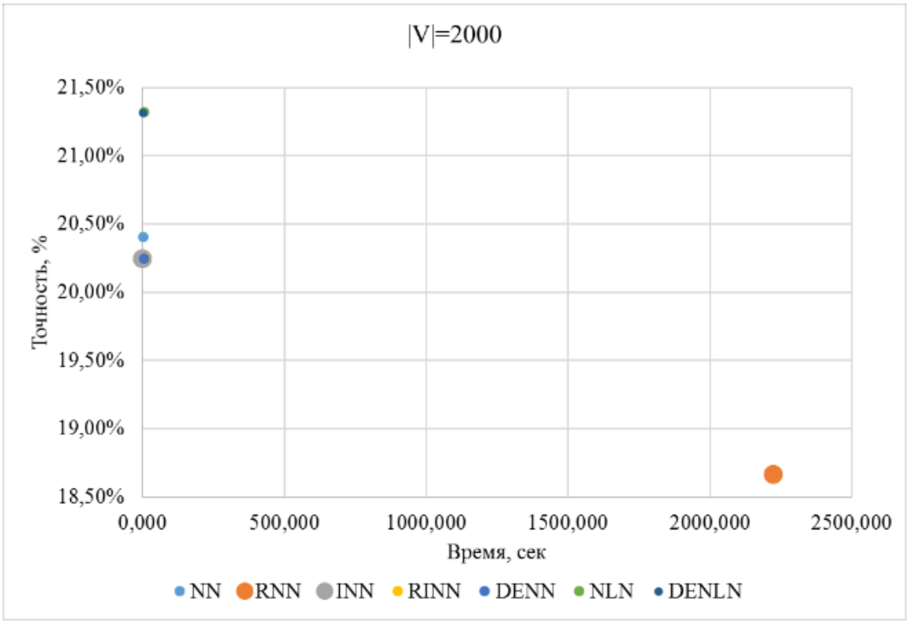


Fig. 10. The results for |V|=2000

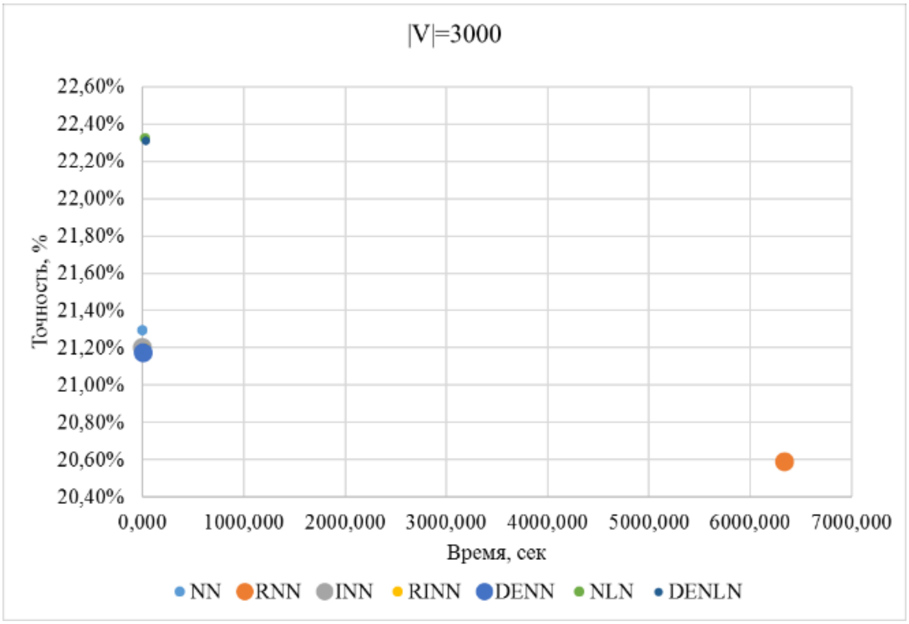


Fig. 11. The results for |V|=3000

References

[1]. C. E. Noon and J. C. Bean, "An Efficient Transformation of The Generalized Traveling Salesman Problem," INFOR Information Systems and Operational Research, vol. 31, no. 1, February 1993.

[2]. H. Thimbleby, "The directed Chinese Postman Problem," Software Practice and Experience, vol. 33, no. 11, pp. 1081-1096, 2003.

[3]. Mei-Ko Kwan, «Graphic programming using odd or even points» Acta Mathematica Sinica, p. 263–266, 1960.

[4]. G. Laporte and M. Blais, "Exact Solution of the Generalized Routing Problem through Graph Transformations," Operations Research, vol. 54, no. 8, pp. 906-910, 2003.

[5]. F. L. Pimentel, «Double-ended nearest and loneliest neighbour–a nearest neighbour heuristic variation for the travelling salesman problem» Revista de Ciências da Computação, т. 6, № 6, 2016.

[6]. P. Vreda and P. Black, Dictionary of Algorithms and Data Structures, National Institute of Standards and Technology, 2014.

[7]. G. Laporte, «Modeling and solving several classes of arc routing problems as traveling salesman problems» Computers & operations research, т. 24, № 11, pp. 1057-1061, 1997.

[8]. S. Bönisch, «Implementierung der Edmonds-Johnson Heuritik für das Mixed Chinese Postman Problem» 21 December 1999.

[9]. Á. Corberán, "Arc Routing Problems: Data Instances," [Online]. Available: http://www.uv.es/corberan/instancias.htm. [Accessed 3 April 2017].

Смешанная задача китайского почтальона

М.К. Горденко <mkgordenko@gmail.ru>
С.М. Авдошин <savdoshin@edu.hse.ru>
Департамент программной инженерии, Национальный исследовательский университет «Высшая школа экономики», 101000, Москва, ул. Мясницкая, д. 20

Аннотация. Задачи маршрутизации важны для областей логистики и управления трансортом. Задачи маршрутизации в основном связаны с определением оптимального набора путей в мультиграфе. Задача китайского почтальона (CPP) является особым случаем задачи маршрутизации, имющим много потенциальных приложений. Мы предлагаем решение MCPP (специального NP-полного случая CPP на смешанном мультиграфе) с использованием редуцирования исходной задачи к обобщенной задаче коммивояжера (General Traveling Salesman Problem, GTSP). Указываются варианты CPP. Представлены математические формулировки некоторых проблем. Показан алгоритм редуцирования MCPP в мультиграфе к GTSP. Приводятся экспериментальные результаты решения MCPP в мультиграфе посредством редуцирования к GTSP.

Ключевые слова: смешанная задача китайского почтальона, задача маршрутизации, эвристический алгоритм, задача коммивояжера

DOI: 10.15514/ISPRAS-2017-29(4)-7

Для цитирования: Горденко М.К., Авдошин С.М. Смешанная задача китайского почтальона. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 107-122 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-7

Список литературы

[1]. C. E. Noon and J. C. Bean, "An Efficient Transformation of The Generalized Traveling Salesman Problem," INFOR Information Systems and Operational Research, vol. 31, no. 1, February 1993.
[2]. H. Thimbleby, "The directed Chinese Postman Problem," Software Practice and Experience, vol. 33, no. 11, pp. 1081-1096, 2003.
[3]. Mei-Ko Kwan, «Graphic programming using odd or even points» Acta Mathematica Sinica, p. 263–266, 1960.
[4]. G. Laporte and M. Blais, "Exact Solution of the Generalized Routing Problem through Graph Transformations," Operations Research, vol. 54, no. 8, pp. 906-910, 2003.
[5]. F. L. Pimentel, «Double-ended nearest and loneliest neighbour–a nearest neighbour heuristic variation for the travelling salesman problem» Revista de Ciências da Computação, т. 6, № 6, 2016.
[6]. P. Vreda and P. Black, Dictionary of Algorithms and Data Structures, National Institute of Standards and Technology, 2014.
[7]. G. Laporte, «Modeling and solving several classes of arc routing problems as traveling salesman problems» Computers & operations research, т. 24, № 11, pp. 1057-1061, 1997.
[8]. S. Bönisch, «Implementierung der Edmonds-Johnson Heuritik für das Mixed Chinese Postman Problem» 21 December 1999.
[9]. Á. Corberán, "Arc Routing Problems: Data Instances". http://www.uv.es/corberan/instancias.htm. [Дата обращения 03.04.2017].

The Metric Travelling Salesman Problem: The Experiment on Pareto-optimal Algorithms

S.M. Avdoshin <savdoshin@hse.ru>
E.N. Beresneva <katrinberesneva@yandex.ru>
Department of Software Engineering,
National Research University Higher School of Economics,
20, Myasnitskaya st., Moscow, 101000 Russia

Abstract. The Metric Travelling Salesman Problem is a subcase of the Travelling Salesman Problem (TSP), where the triangle inequality holds. It is a key problem in combinatorial optimization. Solutions of the Metric TSP are generally used for costs minimization tasks in logistics, manufacturing, genetics and other fields. Since this problem is NP-hard, heuristic algorithms providing near optimal solutions in polynomial time will be considered instead of the exact ones. The aim of this article is to experimentally find Pareto optimal heuristics for Metric TSP under criteria of error rate and run time efficiency. Two real-life kinds of inputs are intercompared - VLSI Data Sets based on very large scale integration schemes and National TSPs that use geographic coordinates of cities. This paper provides an overview and prior estimates of seventeen heuristic algorithms implemented in C++ and tested on both data sets. The details of the research methodology are provided, the computational scenario is presented. In the course of computational experiments, the comparative figures are obtained and on their basis multi-objective optimization is provided. Overall, the group of Pareto-optimal algorithms for different $\boldsymbol{N}$ consists of some of the MC, SC, NN, DENN, CI, GRD, CI + 2-Opt, GRD + 2-Opt, CHR and LKH heuristics.

Keywords: metric travelling salesman problem, heuristic algorithms, Pareto-optimality

DOI: 10.15514/ISPRAS-2017-29(4)-8

For citation: Avdoshin S.M., Beresneva E.N. The Metric Travelling Salesman Problem: Pareto-optimal Algorithms. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017. pp. 123-138. DOI: 10.15514/ISPRAS-2017-29(4)-8

1. Introduction

The Travelling Salesman Problem (TSP) is one of the most widely known questions in a class of combinatorial optimization problems. Essentially, to meet a challenge of the TSP is to find a Hamiltonian circuit of minimal length. A subcase of the TSP is Metric TSP where all of the edge costs are symmetric, and they satisfy the triangle inequality.

The methods for solving the TSP have been developed for many years, and since the problem is NP-hard, it continues to be topical. The TSP has seen applications in the areas of logistics, genetics, manufacturing, telecommunications and neuroscience [1]. The most common practical interpretation of the TSP relates to the movement of people and vehicles around tours, such as searching for the shortest tour through N cities, school bus route planning, and postal delivery. In addition, the TSP plays an important role in very large-scale integration (VLSI).

The purpose of this study is to determine the group of Pareto-optimal algorithms among the set of selected ones for Metric TSP by criteria of run time and qualitative performance.

Clearly, a study of this type is inevitably restricted by various constraints, in this research only heuristic algorithms constructing near optimal solutions in polynomial time will be considered instead of the exact ones.

The paper is structured as follows. First, the theoretical basis is described. It presents definition of resource-efficient parameters, Pareto optimization and, at last, the formulation of the aim of the project. Then the description of methods to be used is provided with their prior estimates. After that the details of the research methodology and expected results are mentioned.

2. Theoretical basis

In this paper, mathematical formulation of Metric TSP is adopted as stated here [2].

2.1 Parameters for Pareto-optimality

Let M be a set of selected heuristic algorithms for Metric TSP. There are two parameters of resource-efficiency for $m \in M$ for each number of vertices N in data set:

- $f_\varepsilon(m, N)$ — qualitative performance;
- $f_t(m, N)$ — running time.

Qualitative performance can be calculated using:

$$f_\varepsilon(m, N) = \frac{f(s) - f(s_0)}{f(s_0)} * 100\%,$$

where $f(s)$ is the obtained tour length and $f(s_0)$ is the optimal tour length. The values of optimal tour lengths are taken from the open libraries VLSI Data Sets and National TSPs as the lengths of the best found (exactly) or reported solutions for each of the instances [3] [4].

2.2 Pareto-optimality

The algorithm $m_0 \in M$ is said to be Pareto optimal if $(\forall m \in M)\Big((m \neq m_0) \Rightarrow \big(f_\varepsilon(m) > f_\varepsilon(m_0)\big) \vee \big(f_t(m) > f_t(m_0)\big)\Big)$.

2.3 The aim of the study

The aim is to find a set $M_0 = \Big\{(\forall m \in M)\Big((m \neq m_0) \Rightarrow \big(f_\varepsilon(m) > f_\varepsilon(m_0)\big) \vee \big(f_t(m) > f_t(m_0)\big)\Big)\Big\}$ of Pareto-optimal algorithms for Metric TSP by criteria of time and qualitative performance.

3. Algorithms

Algorithms for solving the TSP may be divided into two classes:

- Exact algorithms, and
- Heuristic (or approximate) algorithms.

Exact algorithms are aimed at finding optimal solutions. However, a major drawback is connected with their time efficiency. It is a common knowledge that there are no exact algorithms running in polynomial time. Thus, only small datasets can be solved in reasonable time. For example, the 4410-vertex problem is believed to be the largest Metric TSP ever solved with respect to optimality [3].

In this paper, some algorithms from a class of heuristic search algorithms will be taken into account. They are designed to run quickly and to get an approximate solution to a given problem.

Heuristic algorithms are subdivided into two groups. The first group includes tour construction algorithms that have one feature in common — the tour is built by adding a new vertex at each step. The second group consists of tour-improving algorithms that, according to Applegate, '…take as input an approximate solution to a problem and attempt to iteratively improve it by moving to new solutions that are close to the original'. Full classification of heuristic algorithms has already been presented in [2].

In order to restrict our investigation, it was decided to choose only three types of tour improving algorithms — the most simple local-optimal method (2-Opt), the most perspective one (LKH) and one of the best swarm intelligence methods — algorithm qCABC based on bee colony agents.

The list of used algorithms for Metric TSP is as follows.

3.1 Nearest Neighbour (NN)

The key to NN is to initially choose a random vertex and to add repeatedly the nearest vertex to the last appended, unless all vertices are used [5].

3.2 Double Ended Nearest Neighbour (DENN)

This algorithm is a modification of NN. Unlike NN, not only the last appended vertex is taken into consideration, so the closest vertex to both of endpoints in the tour is added [6].

3.3 Greedy (GRD)

The Greedy heuristic constructs a path by adding the shortest edge to the tour until a cycle with K edges, $K < N$, is created, or the degree of any vertex exceeds two [7].

3.4 Nearest Addition (NA)

The fundamental idea of NA is to start with an initial subtour made of the shortest edge and to add repeatedly other vertices which are the closest to the vertices being already in the cycle. It should be noted that insertion place is not specially calculated. It is always added after the nearest vertex in the cycle. Algorithm is terminated when all vertices are used and inserted in the tour.

3.5 Nearest Insertion (NI), Cheapest Insertion (CI), Farthest Insertion (FI), Arbitrary Insertion (AI), Nearest Segment Insertion (NSI)

The start step of these algorithms is similar to NA (except for FI, where the longest edge is found). Next, other vertices are added repeatedly using various rules. Depending on the algorithm the vertex not yet in the cycle should be inserted so that:

- In NI it is the closest to any node in the tour;
- In CI its addition to the tour gives a minor increment of its length;
- In FI it is the farthest to any node in the cycle;
- IN AI it is the random vertex not yet in the cycle;
- In NSI distance between the node and any edge in the tour is minimal.

The previous step should be repeated until all vertices are added to the cycle.

The feature of these methods is additional computation that selects the best place for each inserting node [6] [8].

3.6 Double Minimum Spanning Tree (DMST)

DMST method is based on the construction of a minimal spanning tree (MST) from the set of all vertices. After MST is built, the edges are doubled in order to obtain an Eulerian cycle, containing each vertex at least once. Finally, a Hamiltonian circuit is made from an Eulerian circuit by sequential (or greedy) removing occurrences of each node [9].

3.7 Double Minimum Spanning Tree Modified (DMST-M)

This algorithm is a modification of DMST. Unlike DMST, it is necessary to remove duplicate nodes from an Eulerian cycle using triangle inequality instead of greedy method.

3.8 Christofides (CHR)

This method is a modification of DMST that was proposed by Christofides [10]. The difference between CHR and DMST is addition of minimum weight matching calculation to the first algorithm.

3.9 Moore Curve (MC)

This is a recursive geometric method. Vertices are sorted by the order they are located on the plane. Only the two-dimensional example of Moore curve is implemented [11]. Figure 1 shows the order of the cells after one, two and three subdivision steps respectively [11].

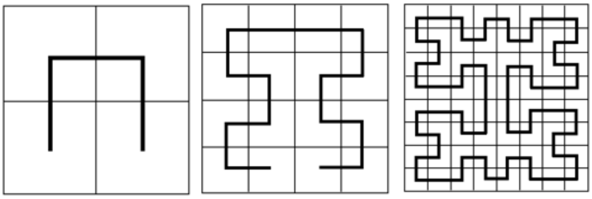


Fig. 1. The order for the Moore curve after 1, 2 and 3 subdivision steps

3.10 Sierpinski Curve (SC)

This algorithm is also included in the family of Space-Filling Curves combinatorial algorithms as MC. SC is more symmetric than MC [12]. Figure 2 shows the order of the cells after one, two and three subdivision steps respectively.

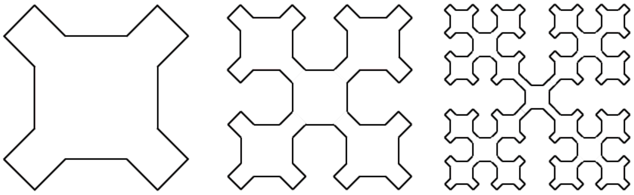


Fig. 2. The order for the Sierpinski curve after 1, 2 and 3 subdivision steps

3.11 2-Opt

The main idea behind 2-Opt is to take a tour that has one or more self-intersections and to remove them repeatedly. In mathematical terms, edges ab and cd should be deleted and new edges ac and bd should be inserted, if $d(a,b) + d(c,d) > d(a,c) + d(b,d)$ [13].

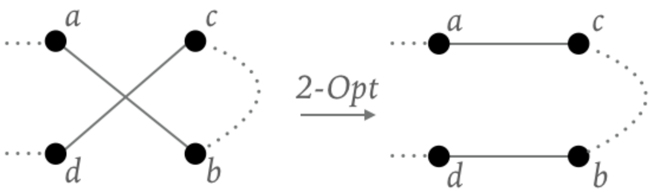


Fig. 3. 2-Opt modification

3.12 Helsgaun's Lin and Kernighan Heuristic (LKH)

LKH uses the principle of 2-Opt algorithm and generalizes it. In this heuristic, the k-Opt, where $k = \overline{2..\sqrt{N}}$, is applied, so the switches of two or more edges are made in order to improve the tour. This method is adaptive, so decision about how many edges should be replaced is taken at each step [14].

It should be noted that because of complexity of LKH algorithm, it was not implemented by the authors of research. The original open source code [15] was used to carry out experiments. All the parameters were not changed, so they were used by default.

3.13 Quick Combinatorial Artificial Bee Colony (qCABC).

This is one of the Swarm Intelligence methods, which is based on colony of bees. Algorithm suggests that all agents are divided into the three groups: scout bees (looking for new random solutions), employed bees (keeping information and sharing it) and onlooker bees (choosing the solution to explore) [16].

3.14 Estimates

Estimated upper bounds for the algorithms can be calculated as are the ratio of $\frac{f(s)}{f(s_0)}$. According to [1], for any k-Opt algorithm, where $k \leq N/4$, problems may be constructed such that the error is almost 100%. So 2-Opt and LKH algorithms have approximate upper bound 2. Upper-bound estimates and running times of the algorithms are represented in Table 1.

Table 1. Upper-bound estimates and running time of algorithms

#	Algorithm	Upper-bound estimate	Running time
1	NN	$0.5\lceil\log_2 N + 1\rceil$	$O(N^2)$
2	DENN		
3	GRD		$O(N^2 \log N)$
4	NA	$2 - \frac{2}{N}$	$O(N^2)$
5	NI		
6	CI		$O(N^2 \log N)$
7	FI		$O(N^2)$
8	AI		
9	NSI		
10	DMST		
11	DMST-M		
12	CHR	$\frac{3}{2} - \frac{1}{N}$	$O(N^3)$
13	2-Opt	≈ 2	$O(N^2)$
14	LKH		$O(N^{2.2})$
15	MC	$\log N$	$O(N \log N)$
16	SC		
17	qCABC	?	$O(N^2)$

4. Experimental research

This section documents details of the research methodology. The experiment is carried out on a 1.3 GHz Intel Core i5 MacBook Air. It includes the qualitative performance and the run time efficiency of the current implementations.

Heuristics are implemented in C++. Two types of data bases from an open library TSPLIB are selected. The first one is VLSI data sets [3]. There are 102 instances in the VLSI collection that range in size from 131 vertices up to 744,710 vertices. All of these instances are tested. The first dataset is National TSPs, which includes 25 instances that vary from 29 to 71009 points [4].

There is one data set for each number of vertices for all input data. The integer Euclidean metric distance is used, so coordinates of nodes and distances between them have integer values. The distance d between some nodes v and w is calculated as follows:

$$d(v,w) = \left\lfloor \sqrt{|x(v) - x(w)|^2 + |y(v) - y(w)|^2} + 0.5 \right\rfloor$$

The computational experiment corresponds to the following scenario:

```
Input: Algorithms, input datasets (VLSI Data Sets, National TSPs)
    1:  foreach tour construction and composite algorithm m
    2:      foreach tour improving algorithm m’
    3:          foreach dataset type DT from input datasets
    4:              foreach dataset D form DT
```

```
5:                    for i ∈ {1…11} // tour construction stage
6:                        solution s = run algorithm m on D
7:                        if (i > 1)
8:                            calculate $f_{\varepsilon_i}(m, D)$, $f_{t_i}(m, D)$
9:                   calculate $f_{\varepsilon_{min}}(m, D)$
10:            remember best solution $s_0$
11:            calculate $\sigma\left(\sum_{i=1}^{10} f_{t_i}(m, D)\right)$
12:            calculate $f_{t_{avg}}(m, D) = \frac{f_{t_1}(m,D)+\cdots+f_{t_{10}}(m,D)}{10}$
13:            if (m is composite) continue
14:            for i ∈ {1…11} // improvement stage on $s_0$
15:                solution s = run algorithm m’ on D
16:                if (i > 1)
17:                    calculate $f_{\varepsilon_i}(s_0 + m', D)$, $f_{t_i}(s_0 + m', D)$
18:            calculate $f_{\varepsilon_{min}}(s_0 + m', D)$
19:            calculate $\sigma\left(\sum_{i=1}^{10} f_{t_i}(s_0 + m', D)\right)$
20:            calculate $f_{t_{avg}}(s_0 + m', D) = \frac{f_{t_1}(s_0+m',D)+\cdots+f_{t_{10}}(s_0+m',D)}{10}$
21:        calculate $E\left(f_{\varepsilon_{min}}(m, D)\right)$, $E\left(f_{\varepsilon_{min}}(s_0 + m', D)\right)$ for all $D$
22:        calculate $\sigma\left(f_{\varepsilon_{min}}(m, D)\right)$, $\sigma\left(f_{\varepsilon_{min}}(s_0 + m', D)\right)$ for all $D$
23:        calculate $max\left(f_{\varepsilon_{min}}(m, D)\right)$, $max\left(f_{\varepsilon_{min}}(s_0 + m', D)\right)$ for all $D$
24:        calculate $min\left(f_{\varepsilon_{min}}(m, D)\right)$, $min\left(f_{\varepsilon_{min}}(s_0 + m', D)\right)$ for all $D$
```

Fig. 3. Computational scenario

Metrics used in scenario have following meanings:

- $f_{\varepsilon_i}(m, D)$ — qualitative performance of m (one iteration),
- $f_{t_i}(m, D)$ — running time of m (one iteration),
- $f_{\varepsilon_{min}}(m, N)$ — best qualitative performance of m,
- $f_{t_{avg}}(m, N)$ — average running time of m (sec),
- $\sigma\left(\sum_{i=1}^{10} f_{t_i}(m, D)\right)$ — standard deviation of running time estimates through 10 iterative runs,
- $E\left(f_{\varepsilon_{min}}(m, N)\right)$ — expected value of qualitative performance of m for one DT,
- $\sigma\left(f_{\varepsilon_{min}}(m, N)\right)$ — standard deviation of qualitative performance of m for one DT,
- $max\left(f_{\varepsilon_{min}}(m, N)\right), min\left(f_{\varepsilon_{min}}(m, N)\right)$ — maximum and minimum values of qualitative performance of m for one DT.

Qualitative performance metrics are represented in Table 2. Table color scheme varies from green (the best result in a column) to red (the worst value in a column).

Table 2. Running time of algorithms

Algorithms	$E(f_\varepsilon)$	**max** f_ε	**min** f_ε	$\sigma(f_\varepsilon)$
LKH	0,08%	0,23%	0,00%	0,07%
CHR + 2-Opt	6,14%	12,14%	3,47%	1,59%
GRD + 2-Opt	6,79%	10,82%	4,69%	1,57%
DENN + 2-Opt	11,06%	22,26%	4,39%	5,30%
NN + 2-Opt	11,89%	24,91%	3,90%	2,36%
CHR	12,60%	16,82%	9,31%	1,41%
CI + 2-Opt	13,04%	21,86%	6,74%	2,83%
NI + 2-Opt	14,60%	29,66%	5,86%	6,33%
DMST-M + 2-Opt	16,08%	35,78%	4,80%	9,29%
GRD	17,31%	31,34%	10,30%	3,83%
NSI + 2-Opt	17,63%	33,65%	8,92%	6,87%
DMST + 2-Opt	19,08%	39,12%	6,91%	10,52%
CI	20,28%	25,05%	12,46%	1,96%
DENN	22,82%	33,38%	11,97%	2,47%
NN	25,38%	32,68%	13,94%	2,62%
NA + 2-Opt	27,04%	57,90%	6,79%	17,84%
NI	28,07%	35,29%	14,89%	2,87%
FI + 2-Opt	28,90%	58,05%	4,01%	17,68%
DMST-M	32,46%	41,68%	18,55%	4,32%
SC + 2-Opt	36,20%	166,45%	8,11%	31,69%
NSI	36,23%	48,17%	19,15%	5,46%
MC + 2-Opt	36,47%	177,83%	6,21%	39,70%
DMST	40,09%	48,88%	33,16%	3,07%
AI + 2-Opt	50,23%	77,85%	5,26%	23,43%
NA	51,30%	59,23%	35,38%	4,67%
FI	56,88%	66,09%	31,59%	5,98%
MC	64,49%	242,41%	33,07%	41,08%
SC	66,16%	246,64%	30,76%	42,13%
AI	85,20%	100,92%	65,78%	6,81%

The time limit on algorithm's running time is introduced. It is 11 800 seconds ≈ 3 hours and 20 minutes, at the maximum. That means computational time for one experiment cannot exceed 11 800 seconds * 11 runs ≈ 36 hours ≈ 1.5 days.

5. Results

Experimental results showed that algorithm qCABC takes a large amount of time (more than 'the slowest' CHR) and gives improvement in accuracy even less than 'the most rough' 2-Opt. So qCABC as tour improving algorithm is admitted to be "unviable".

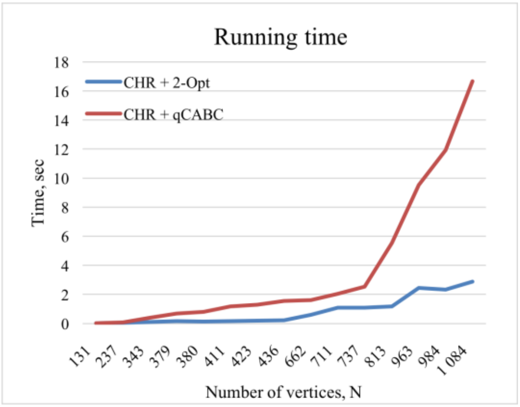


Fig. 4. Running time comparison of CHR + 2-Opt and CHR + qCABC algorithms.

We decided to select 10 pairs of data sets from VLSI and National TSPs with similar number of vertices (see Table 3) to plot charts that illustrate Paretos.

Table 3. Pairs of input datasets from VLSI and National TSPs

	VLSI	**National TSP**
Number of vertices, N	737	734
	984	980
	1 973	1 979
	3 386	3 496
	7 168	7 146
	10150	9 976
	14 233	14 185
	16 928	16 862
	22 777	22 775
	33 203	33 708

The charts for pair with $N = 22\ 775$ and $N = 22\ 777$ are shown below (see Fig. 5, Fig. 6, Fig. 7, Fig. 8). The name of each TSPLIB instance is shown in chart title. The horizontal axis represents the time performance of methods in seconds. The vertical axis shows the gap between optimal and obtained solutions, expressed in percent. Pareto-optimal methods are highlighted in red. The points which are represented by Pareto solutions are bigger than non-Pareto-optimal solutions.

There are two charts (see Fig. 6, Fig. 8) where not all algorithms are compared. These auxiliary charts are enlarged copies of their originals. Their role is to graphically illustrate Pareto-optimal algorithms at scale-up.

Results on VLSI Data sets only are reported in more detail in [2].

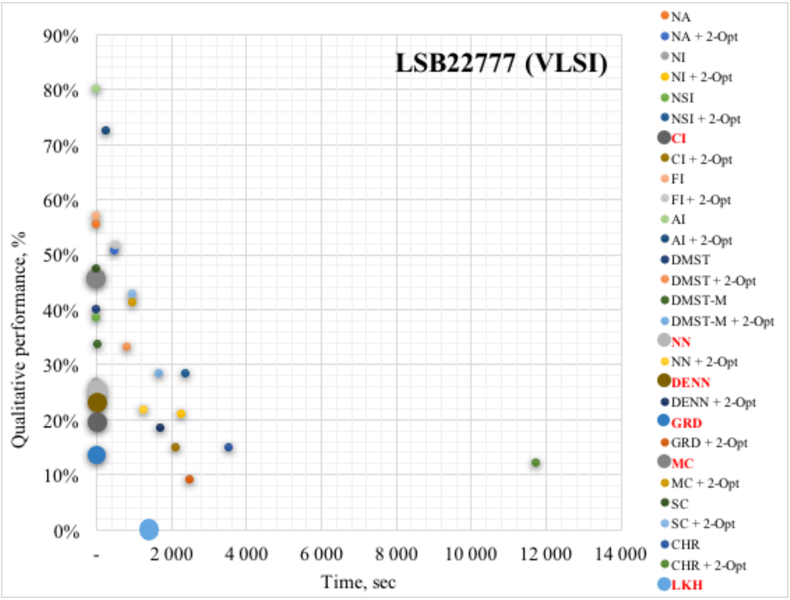


Fig. 5. Pareto-optimal algorithms for LSB22777.tsp (N = 22777)

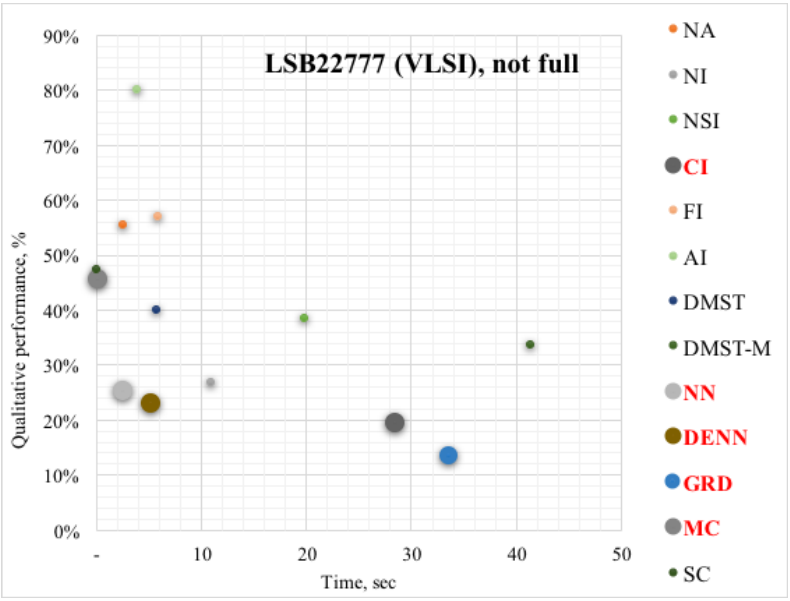


Fig. 6. Pareto-optimal algorithms for LSB22777.tsp (N = 22777), scaled-up

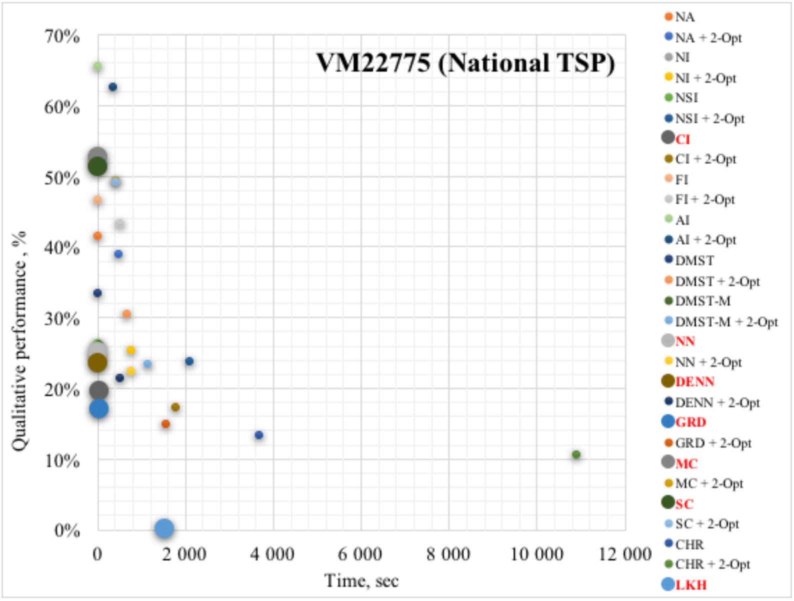


Fig. 7. Pareto-optimal algorithms for VM22775.tsp (N = 22775)

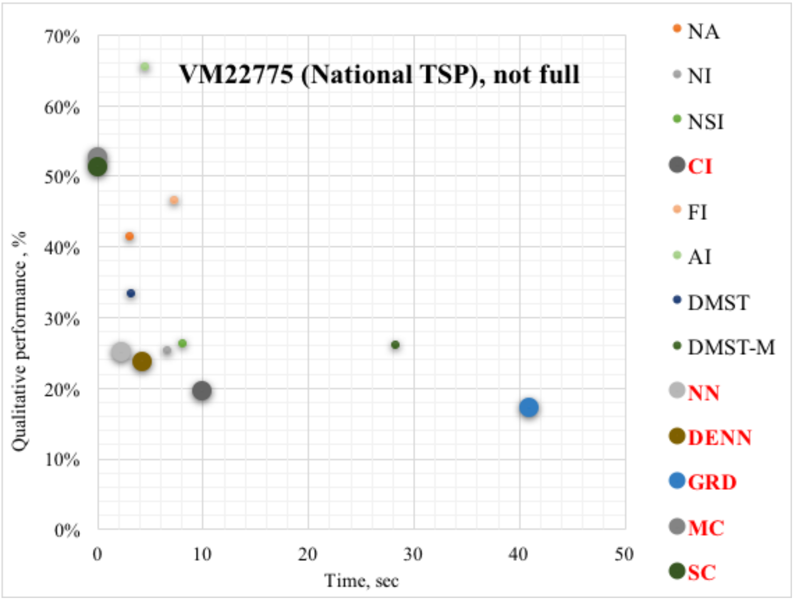


Fig. 8. Pareto-optimal algorithms for VM22775.tsp (N = 22775), scaled-up

Pareto-optimal solutions, that can be suggested on the basis of both data sets only, are shown in Table 4 and they are sorted in the order of increase of running time:

- Moore Curve (MC);
- Sierpinski Curve (SC) — this algorithm depends on type of input data, so qualitative performance estimates are unstable;
- Nearest Neighbour (NN);
- Double Ended Nearest Neighbour (DENN);
- Cheapest Insertion (CI) is Pareto-optimal if $N \lesssim 400\,000$ because of introduced time limit; if $N \lesssim 3\,500$ CI's behavior fluctuates;
- Greedy (GRD) — is Pareto-optimal if $N \lesssim 30\,000$ because of memory limits — $\frac{N(N-1)}{2}$ pairs of edges are needed to be kept simultaneously ;
- Cheapest Insertion and 2-Opt (CI + 2-Opt) — is Pareto-optimal if $30\,000 \lesssim N \lesssim 100\,000$;
- Greedy and 2-Opt (GRD + 2-Opt) — is Pareto-optimal if $N \lesssim 800$;
- Christofides (CHR) — is Pareto-optimal if $N \lesssim 2\,000$;
- Helsgaun's Lin and Kernighan Heuristic (LKH) — this algorithm works excellent if $N \lesssim 55\,000$, however if input data size exceeds $55\,000$ than time limit is met.

Table 4. Pairs of input datasets from VLSI and National TSPs

Algorithm	**Number of vertices, *N* (thousands)**							
	(0; 0.8)	[0.8; 2)	[2; 3.5)	[3.5; 30)	[30; 55)	[55; 100)	[100; 400)	[400; 700)
MC	+	+	+	+	+	+	+	+
SC	±	±	±	±	±	±	±	±
NN	+	+	+	+	+	+	+	+
DENN	+	+	+	+	+	+	+	+
CI	±	±	±	+	+	+	+	
GRD	+	+	+	+				
CI + 2-Opt					+	+		
GRD + 2-Opt	+							
CHR	+	+						
LKH	+	+	+	+	+			

The "+" sign means that the algorithm in the same row is supposed to be Pareto-optimal at the range of vertices defined in the same column. The "±" sign shows that experiments did not clearly define if it is Pareto-optimal or not.

6. Conclusion

The presented study is undertaken to determine what heuristics for Metric TSP should be used in specific circumstances with limited resources.

This paper provides an overview of seventeen heuristic algorithms implemented in C++ and tested on both the VLSI data set and instances of National TSPs. In the course of computational experiments, the comparative figures are obtained and on their basis multi-objective optimization is provided. Overall, the group of Pareto-optimal algorithms for different *N* consists of some of the MC, SC, NN, DENN, CI, GRD, CI + 2-Opt, GRD + 2-Opt, CHR and LKH heuristics.

In our future work, we are going to fine-tune parameters of LKH method using genetic algorithms of search optimization. Further, it is possible to increase the number of heuristic algorithms, to transit to other types of test data and to conduct experiments using different metrics in order to ensure that a Pareto optimal group is sustainable.

The practical applicability of our findings is to present Pareto optimal algorithms that lead to solutions with maximum accuracy under the given resource limitations. The results can be used for scientific purposes by other researchers and for cost minimization tasks.

References

[1]. Applegate, D., Bixby, R., Chvatal, V., Cook, W. The Traveling Salesman Problem: A Computational Study, Princeton: Princeton University Press, 2011.

[2]. Beresneva (Chirkova), E.N., Avdoshin, S.M. Pareto-optimal Algorithms for Metric TSP: Experimental Research. International Journal of Open Information Technologies , vol. 5, № 5, pp. 16-24, 2017, ISSN: 2307-8162.

[3]. Department of Combinatorics and Optimization at the University of Waterloo. Status of VLSI Data Sets. University of Waterloo (web-site). Available at: http://www.math.uwaterloo.ca/tsp/vlsi/summary.html, accessed 29.04.2017.

[4]. University of Waterloo. National Travelling Salesman Problems. UWaterloo, (web-site). Available at: http://www.math.uwaterloo.ca/tsp/world/countries.html, accessed 16.04.2017.

[5]. Flood, M.M. The traveling-salesman problem. Operation research, vol. 4, pp. 61-75, 1956.

[6]. Rosenkrantz, D. Stearns, R., Lewis II, P. An analysis of several heuristics for the traveling salesman problem, vol. 6, pp. 563-581, 1977.

[7]. Cook, W.J. Combinatorial optimization, New York: Wiley, 1998.

[8]. Hahsler, M., Hornik, K. TSP — Infrastructure for the Traveling Salesperson Problem, vol. 23, № 2, 2007.

[9]. Christofides, N. Graph theory — An Algorithmic Approach, New York: Academic Press, 1974.

[10]. Christofides, N. Worst-case analysis of a new heuristic for the travelling salesman problem. Graduate School of Industrial Administration, CMU, 1976.

[11]. Buchin, K. Space-Filling Curves. Organizing Points Sets: Space-Filling Curves, Delaunay Tessellations of Random Point Sets, and Flow Complexes, Berlin, Free University of Berlin, 2007, pp. 5-30.

[12]. Bartholdi, J., Platzman, L., Collins R., Warden, W. A Minimal Technology Routing System for Meals on Wheels, vol. 13, № 3, pp. 1-8, 1983.

[13]. Aarts, E., Lenstra, J. Local Search in Combinatorial Optimization, Princeton, New Jersey: Princeton University Press, 2003.

[14]. Helsgaun, K. An effective implementation of the Lin–Kernighan traveling salesman heuristic, EJOR, vol. 12, pp. 106-130, 2000.
[15]. Helsgaun, K. LKH (web-site). Available at: http://www.akira.ruc.dk/~keld/research/LKH, accessed 24.02.2017.
[16]. Gorkemli, B., Karaboga, D. Quick Combinatorial Artificial Bee Colony -qCABC- Optimization Algorithm for TSP, vol. 1, № 5, 2013.

Метрическая задача коммивояжера: экспериментальное исследование Парето-оптимальных алгоритмов

С.М. Авдошин <savdoshin@hse.ru>
Е.Н. Береснева <katrinberesneva@yandex.ru>
Департамент программной инженерии,
Национальный исследовательский университет "Высшая школа экономики",
101000, Россия, г. Москва, ул. Мясницкая, д. 20.

Аннотация. Задача коммивояжера — одна из важнейших задач теории графов и комбинаторной оптимизации, суть которой состоит в нахождении гамильтонова цикла наименьшей длины. Разработка методов для решения задачи коммивояжера осуществляется на протяжении многих лет, и, по-прежнему, остается актуальной, поскольку задача является NP-трудной. Решения применяются, в основном, для минимизации производственных и логистических затрат и издержек. В работе рассматривается частный случай общей постановки задачи коммивояжера, в котором выполняется свойство метрики — метрическая задача коммивояжера. Целью данной работы является определение группы Парето оптимальных алгоритмов решения метрической задачи коммивояжера по критериям времени работы и точности решения в ходе экспериментального исследования. В связи с тем, что задача коммивояжера является NP-трудной, в работе рассматриваются только эвристические алгоритмы, позволяющие получить приближенные решения за полиномиальное время. В статье представлено краткое описание используемых алгоритмов решения метрической задачи коммивояжера, указаны их априорные точностные и временные оценки. Приведено описание плана эксперимента. Данными для экспериментального исследования послужили два набора из открытой библиотеки данных для метрической задачи коммивояжера, основанные на высоко-интегральных вычислительных схемах (VLSI Data Sets) и географических координатах (высоте и широте) городов в различных странах (National TSPs). В результате исследований выявлены оптимальные по Парето алгоритмы для наборов данных различных размерностей — до 700 тысяч вершин. Для каждого N в число Парето-оптимальных алгоритмов входят некоторые из алгоритмов MC, SC, NN, DENN, CI, GRD, CI + 2-Opt, GRD + 2-Opt, CHR и LKH. Приведена таблица, содержащая информацию о результатах экспериментов.

Ключевые слова: метрическая задача коммивояжера, эвристический алгоритм, оптимальность по Парето.

DOI: 10.15514/ISPRAS-2017-29(4)-8

Для цитирования: Авдошин С.М., Береснева Е.Н. Метрическая задача коммивояжера: экспериментальное исследование Парето-оптимальных алгоритмов. *Труды ИСП РАН*, том 29, вып. 4, 2017 г. стр. 123-138 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-8

Список литературы

[1]. Applegate, D., Bixby, R., Chvatal, V., Cook, W. The Traveling Salesman Problem: A Computational Study, Princeton: Princeton University Press, 2011.
[2]. Beresneva (Chirkova), E.N., Avdoshin, S.M. Pareto-optimal Algorithms for Metric TSP: Experimental Research. International Journal of Open Information Technologies , 5, № 5, pp. 16-24, 2017, ISSN: 2307-8162.
[3]. Department of Combinatorics and Optimization at the University of Waterloo. Status of VLSI Data Sets. University of Waterloo (web-site). Доступно по ссылке: http://www.math.uwaterloo.ca/tsp/vlsi/summary.html, дата обращения 29.04.2017.
[4]. University of Waterloo. National Travelling Salesman Problems. UWaterloo, (web-site). Доступно по ссылке: http://www.math.uwaterloo.ca/tsp/world/countries.html, дата обращения 16.04.2017.
[5]. Flood, M.M. The traveling-salesman problem. Operation research, vol. 4, pp. 61-75, 1956.
[6]. Rosenkrantz, D. Stearns, R., Lewis II, P. An analysis of several heuristics for the traveling salesman problem, 6, pp. 563-581, 1977.
[7]. Cook, W.J. Combinatorial optimization, New York: Wiley, 1998.
[8]. Hahsler, M., Hornik, K. TSP — Infrastructure for the Traveling Salesperson Problem, 23, № 2, 2007.
[9]. Christofides, N. Graph theory — An Algorithmic Approach, New York: Academic Press, 1974.
[10]. Christofides, N. Worst-case analysis of a new heuristic for the travelling salesman problem. Graduate School of Industrial Administration, CMU, 1976.
[11]. Buchin, K. Space-Filling Curves. Organizing Points Sets: Space-Filling Curves, Delaunay Tessellations of Random Point Sets, and Flow Complexes, Berlin, Free University of Berlin, 2007, pp. 5-30.
[12]. Bartholdi, J., Platzman, L., Collins R., Warden, W. A Minimal Technology Routing System for Meals on Wheels, 13, № 3, pp. 1-8, 1983.
[13]. Aarts, E., Lenstra, J. Local Search in Combinatorial Optimization, Princeton, New Jersey: Princeton University Press, 2003.
[14]. Helsgaun, K. An effective implementation of the Lin–Kernighan traveling salesman heuristic, EJOR, 12, pp. 106-130, 2000.
[15]. Helsgaun, K. LKH (web-site). Доступно по ссылке: http://www.akira.ruc.dk/~keld/research/LKH, дата обращения 24.02.2017.
[16]. Gorkemli, B., Karaboga, D. Quick Combinatorial Artificial Bee Colony -qCABC- Optimization Algorithm for TSP, 1, № 5, 2013.

Минимизация автоматов с таймаутами и временными ограничениями

А.С. Твардовский <tvardal@mail.ru>
Н.В. Евтушенко <nyevtush@gmail.com>
М.Л. Громов <maxim.leo.gromov@gmail.com>
Национальный исследовательский Томский государственный университет, 634050, Россия, г. Томск, пр. Ленина, 36

Аннотация. Конечные автоматы широко используются для анализа и синтеза управляющих систем. При описании систем, поведение которых зависит от времени, конечный автомат расширяется введением временных аспектов и вводится понятие временного автомата. В настоящей работе мы рассматриваем проблему минимизации автоматов с таймаутами и временными ограничениями, поскольку сложность многих задач в теории автоматов существенно зависит от размеров исследуемой системы. Поведение временного автомата может быть достаточно точно описано соответствующим конечным автоматом, и предлагаемый метод минимизации числа состояний системы основан на использовании такой конечно автоматной абстракции. Более того, далее мы минимизируем и временные аспекты автоматного описания, сокращая продолжительность таймаутов и число переходов с временными ограничениями. Мы также показываем, что для полностью определённого детерминированного временного автомата существует единственная минимальная (каноничная) форма, т. е. единственный приведённый по состояниям и временным аспектам автомат с таймаутами и временными ограничениями, поведение которого совпадает с исходным временным автоматом; например, такая минимальная форма может быть использована при построении проверяющих тестов для проверки функциональных и нефункциональных требований к тестируемой реализации. Предложенный метод к минимизации временных аспектов на основе конечно автоматной абстракции может быть применён и для частных случаев рассматриваемой модели, т. е. для минимизации детерминированных полностью определенных автоматов только с таймаутами или только с временными ограничениями.

Ключевые слова: временные автоматы; приведённая форма; минимальная форма

DOI: 10.15514/ISPRAS-2017-29(4)-8

Для цитирования: Твардовский А.С., Евтушенко Н.В., Громов М.Л. Минимизация автоматов с таймаутами и временными ограничениями. *Труды ИСП РАН,* том 29, вып. 4, 2017 г., стр. 139-154. DOI: 10.15514/ISPRAS-2017-29(4)-8

1. Введение

Конечно-автоматные модели широко используются при анализе и синтезе дискретных систем [1, 2], в частности при реализации компонент управляющих систем [3, 4, 5], причем сложность решения многих задач теории автоматов существенно зависит от числа состояний рассматриваемого автомата. Более того, многие алгоритмы синтеза тестов с гарантированной полнотой [2, 6, 7, 8] для дискретных систем разработаны только для приведённых автоматов, т.е. автоматов, в которых любая пара состояний различима.

При рассмотрении современных систем часто приходится учитывать временные аспекты в их поведении, и соответственно, понятие конечного автомата расширяется введением временных переменных [7, 9, 10]. Известны различные способы добавления временных переменных в описание цифровых компонентов, поведение которых описывается конечными автоматами. В настоящей работе мы рассматриваем автоматы с таймаутами и временными ограничениями [10], которые являются обобщением автоматов только с таймаутами и только с временными ограничениями [11, 12].

Мы предлагаем метод минимизации детерминированного полностью определенного временного автомата, используя, подобно [1], разбиение множества состояний автомата по отношению эквивалентности. По определению, состояния эквивалентны, если автомат в этих состояниях имеет одну и ту же выходную реакцию на любую входную последовательность. Мы вводим класс приведенных по состояниями и временным аспектам автоматов с таймаутами и временными ограничениями, и показываем, что такая минимальная форма является единственной с точностью до изоморфизма. Построение минимальной формы основано на специальной конечно автоматной абстракции, которая сохраняет последовательностные свойства временного автомата. Структура работы следующая. Раздел 2 содержит основные определения и обозначения. В разделе 3 рассматривается алгоритм построения конечно автоматной абстракции и обсуждаются её свойства. В разделе 4 представлен предложенный алгоритм минимизации состояний детерминированного полностью определенного автомата с таймаутами и временными ограничениями. В разделе 5 предлагается алгоритм минимизации временных аспектов автомата с таймаутами и временными ограничениями и показывается, что минимальная форма для такой модели временного автомата единственна.

2. Основные определения и обозначения

Под *конечным автоматом* понимается четвёрка $\mathsf{S} = (S, I, O, h_S)$, где I – множество входных символов, O – *множество выходных символов*, S – конечное непустое множество *состояний*, $h_S \subseteq (S \times I \times O \times S)$ – *отношение переходов*. Соответственно, кортеж (s, i, o, s') описывает переход из состояния

s в состояние s' под действием входного символа i с выдачей выходного символа o. Последовательность пар входной символ/выходной символ называется входо-выходной последовательностью и обозначается α/γ, где α – входная последовательность, и γ – выходная последовательность.

Под *автоматом с таймаутами и временными ограничениями* понимается пятерка $\mathsf{S} = (I, S, O, h_S, \Delta_S)$, где I – входной алфавит, O – выходной алфавит, S – конечное непустое множество состояний, $h_S \subseteq (S \times I \times O \times S \times \Pi \times \mathbb{Z}^+)$ – отношение переходов, Δ_S: $S \rightarrow S \times (\mathbb{N} \cup \{\infty\})$ – функция *таймаута*, определяющая для каждого состояния максимальное время ожидания входного символа, $\mathbb{N}$ – множество натуральных чисел, Π – множество интервалов из промежутка $[0; T)$ вида $\langle a, b\rangle$, где $\langle \in \{[,(\}$, $\rangle \in \{],)\}$, T есть таймаут в текущем состоянии и $\mathbb{Z}^+$ – множество целых неотрицательных чисел. Соответственно, *кортеж* (s, i, o, s', g, d) описывает переход из состояния s в состояние s' под действием входного символа i, поступившего в момент времени t, $t \in g$, после перехода автомата в текущее состояние с выдачей выходного символа o через d тактов времени после поступления входного символа. Иногда время d обработки входного символа называют *выходным таймаутом* или *временем задержки*. Если в некотором состоянии автомата входной сигнал не поступает в течение определенного времени, который превышает (входной) таймаут в текущем состоянии, то автомат может изменить своё состояние. Например, если $\Delta_S(s) = (s', T)$ и в состоянии s в течение T единиц времени на автомат не было подано ни одного входного символа, то автомат переходит в состояние s'. Для таймаута вида $\Delta_S(s) = (s', \infty)$ справедливо, что $s = s'$. После перехода в состояние s' по входному символу либо таймауту отсчет времени начинается с 0. Если $s = s'$, то после достижения таймаута в состоянии s отсчет времени начинается с 0. Функция $time(s, t) = s'$ [7] определяет, в каком состоянии s' находится автомат через t тактов времени, при условии, что входной символ не был подан в течение этого времени. Для временного автомата S через B будем обозначать наибольшую конечную границу для временных интервалов, которая совпадает с максимальной величиной входного таймаута.

Временным входным символом называется пара (i, t), где i – символ входного алфавита, t – время поступления входного символа после выдачи автоматом последнего выходного символа. *Временным выходным символом* называется пара (o, d), где o – символ выходного алфавита, d – число единиц времени между подачей входного символа и выдачей выходного символа. Для временного автомата S и входной временной последовательности $\alpha = (i_1, t_1)$, (i_2, t_2) …, (i_n, t_n) соответствующая выходная временная последовательность $\gamma = (o_1, d_1)$, (o_2, d_2) …, (o_n, d_n) называется (выходной) реакцией. При этом последовательность пар (временной входной символ / временной выходной символ) называется *временной входо-выходной последовательностью* и обозначается α/γ. Для нахождения реакции временного автомата на входной временной символ (i, t) в состоянии s, сначала определяется состояние $s' = time(s, t)$, в котором находится автомат в момент времени t, далее определяется переход (s', i, o, s'', g, d) такой, что ($t - \Sigma$) $\in g$, где Σ есть сумма таймаутов при переходах по таймаутам из состояния s в состояние s'; соответственно, при подаче входного временного символа (i, t) в состоянии s автомат производит выходную реакцию (o, d) и переходит в следующее состояние s''. Таким образом, реакция автомата на временную входную последовательность в состоянии s вычисляется итеративно, начиная с состояния s. Вычитание величины Σ из времени поступления входного символа отражает тот факт, что отсчёт времени «сбрасывается» в 0 при выполнении автоматом переходов по таймауту.

В настоящей работе мы рассматриваем полностью определенные и детерминированные автоматы, т.е. для каждого состояния s и временного входного символа (i, t) единственным образом определяется следующее состояние s'' автомата и выходная реакция (o, d).

На рисунке 1 представлен автомат с таймаутами и временными ограничениями S. Реакции автомата S в состоянии s_0 на временные входные символы (i, 0) и (i, 1) различаются и представляют собой выходные временные символы (o_1, 1) и (o_2, 3) соответственно. Если же входной символ не будет подан на автомат S в состоянии s_0 в течение двух единиц времени, то автомат перейдёт в состояние s_3 по таймауту.

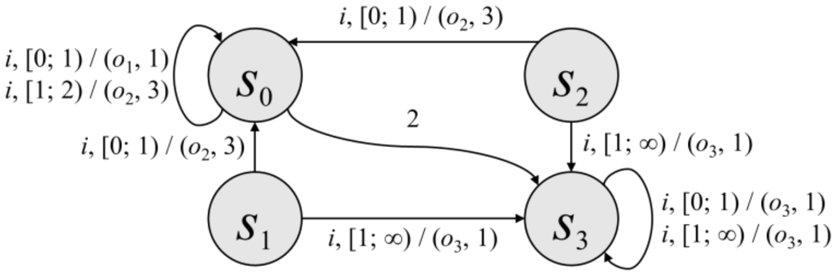


Рис. 1. Временной автомат S
Fig. 1. Timed Finite State Machines S

Состояния s и p полностью определённых детерминированных временных автоматов S и P называются *эквивалентными*, если реакции автоматов в этих состояниях совпадают на любую входную временную последовательность. Если состояния s и p не являются эквивалентными, то они называются *различимыми*. Отношение эквивалентности на множестве состояний автомата индуцирует разбиение на множестве состояний, которое называется *разбиением* по отношению эквивалентности и далее обозначается E. Любые два состояния, принадлежащие одному классу разбиения E, являются эквивалентными; любые два состояния, принадлежащие различным классам разбиения E, являются различимыми.

Два автомата S и P *эквивалентны*, если для каждого состояния автомата S существует эквивалентное состояние в автомате P, и для каждого состояния автомата P существует эквивалентное состояние в автомате S. Автомат называется *приведённым по состояниям*, если любые два состояния в нём различимы. *Приведённой по состояниям формой* временного автомата S называется приведённый автомат, эквивалентный S.

Временные автоматы S и P с одинаковыми входными и выходными алфавитами *изоморфны*, если между множествами состояний и переходов этих автоматов можно установить взаимно однозначное соответствие, т.е. существует взаимно-однозначное отображение H: $S \to P$, такое что $(s_i, i, o, s_j, g, d) \in h_s$, если и только если $(H(s_i), i, o, H(s_j), g, d) \in h_p$ и $\Delta_S(s_i) = (s_j, T)$, если и только если $\Delta_P(H(s_i)) = (H(s_j), T)$. Временной автомат, изоморфный заданному временному автомату, может быть получен переименованием состояний исходного автомата.

3. Конечно автоматная абстракция

Поведение автомата с таймаутами и временными ограничениями в ряде случаев можно адекватно описать при помощи конечного автомата, т.е. с использованием *конечно автоматной абстракции* [10]. В настоящей работе это понятие несколько расширяется для построения конечно автоматной абстракции для временных автоматов с выходными таймаутами. Пусть $\mathsf{S} = (S, I, O, h_S, \Delta_S)$ – детерминированный полностью определенный временной автомат и B – наибольшая конечная граница для временных входных интервалов, в то время как D определяет наибольшую выходную задержку. Построим *конечно автоматную абстракцию временного автомата*, которая является полностью определённым конечным автоматом $\mathsf{A}_\mathsf{S} = (S_A, I \cup \{\mathtt{I}\}, O_A, \lambda_{AS}, s_0)$, где $S_A = \{(s, 0), (s, (0, 1)), \ldots, (s, (B-1, B)), (s, B), (s, (B, \infty)): s \in S\}$, $O_A = \{(o, 0), (o, 1), \ldots, (o, D): o \in O\} \cup \{\mathtt{I}\}$, а $\mathtt{I}$ – специальный символ конечно автоматной абстракции. Для состояния (s, t_j), $t_j = 0, \ldots, B$, автомата A_S и входного символа i, множество λ_{AS} содержит переход $((s, t_j), i, (o, d), (s', 0))$, если и только если существует переход $(s, i, o, s', g_i, d) \in \lambda_S$ такой что $t_j \in g_i$. Для состояния (s, g_j), $g_j = (0, 1) \ldots, (B-1, B), (B, \infty)$, автомата A_S и входного символа i, множество λ_{AS} содержит переход $((s, g_i), i, (o, d), (s', 0))$, если и только если существует переход $(s, i, o, s', g, d) \in \lambda_S$ такой что $g_i \subseteq g$. Переходы по специальному символу $\mathtt{I}$ отражают изменение временной переменной между вещественными значениями из интервала вида (a, b) и целочисленными значениями, либо переход по таймаутам между состояниями. Переходы $((s, n), \mathtt{I}, \mathtt{I}, (s, (n, n+1)))$, $((s, (n-1, n)), \mathtt{I}, \mathtt{I}, (s, n)) \in \lambda_{AS}$, если и только если $n < T < \infty$, где $\Delta_S(s) = (s', T)$. Переход $((s, (n-1, n)), \mathtt{I}, \mathtt{I}, (s', 0)) \in \lambda_{AS}$, если и только если $n = T < \infty$, где $\Delta_S(s) = (s', T)$.

Подобно [10], можно показать, что если исходный временной автомат полностью определённый и детерминированный, то и конечно автоматная абстракция также является полностью определённым детерминированным конечным автоматом.

Временная входная последовательность α может быть переведена в соответствующую последовательность входных символов для конечно автоматной абстракции α_{FSM}. При этом каждый временной входной символ (i, t) представляется последовательностью входных символов абстракции вида $\mathtt{I}$, $\mathtt{I}$, …, $\mathtt{I}$, i, где число символов $\mathtt{I}$ равно числу переходов значений временной переменной из целого числа в элемент интервала вида (a, a + 1) и обратно за время t. В то же время выходная реакция абстракции на последовательность $\mathtt{I}$, $\mathtt{I}$, …, $\mathtt{I}$, i, имеет вид $\mathtt{I}$, $\mathtt{I}$, …, $\mathtt{I}$, (o, d), где число символов $\mathtt{I}$ во входной и выходной последовательности совпадает, а (o, d) есть реакция исходного временного автомата на (i, t). Таким образом, выходная последовательность абстракции γ_{FSM} может быть переведена в соответствующую временную выходную последовательность γ удалением всех символов $\mathtt{I}$.

Утверждение 1. Входо-выходная последовательность α/γ существует во временном автомате S, если и только если входо-выходная последовательность $\alpha_{FSM}/\gamma_{FSM}$ существует в конечно автоматной абстракции A_S.

Доказательство. Рассмотрим состояние s временного автомата S и множество состояний (s, 0), (s, (0, 1)), …, (s, (T – 1, T)) конечно автоматной абстракции A_S, где $\Delta_S(s) = (s', T)$. Каждое состояние A_S соответствует целому моменту времени либо интервалу вида (a, a + 1), в котором находится временная переменная автомата S, а переходы по входному символу $\mathtt{I}$ отражают изменение значения временной переменной автомата S из целого числа в элемент интервала вида (a, a + 1) и обратно. Таким образом, по правилам построения множества переходов абстракции, входо-выходная пара (i, t) / (o, d) существует в автомате S, если и только если существует входо-выходная последовательность $\mathtt{I}$, $\mathtt{I}$, …, $\mathtt{I}$, i / $\mathtt{I}$, $\mathtt{I}$, …, $\mathtt{I}$, (o, d) в автомате A_S, где число символов $\mathtt{I}$ равно числу изменений значений временной переменной из целого числа в элемент интервал вида (a, a + 1) и обратно за время $t < T$. В силу того, что переходу по таймауту $\Delta_S(s) = (s', T)$ автомата S соответствует переход ((s, (T – 1, T)), $\mathtt{I}$, $\mathtt{I}$, (s', 0)) автомата A_S, то аналогичные рассуждения можно провести для состояния s' и множества (s', 0), (s', (0, 1)), …, (s', (T – 1, T)), а также всех последующих состояний, в которые автомат S может перейти по таймауту и соответствующего множества состояний абстракции. В то же время, по правилам построения переходов конечно автоматной абстракции, входной временной символ (i, t) переведёт автомат S в состояние s'', если и только если входная последовательность $\mathtt{I}$, $\mathtt{I}$, …, $\mathtt{I}$, i переведёт автомат A_S в состояние (s'', 0). Далее, для пары состояний s'' и (s'', 0) можно провести аналогичные рассуждения.

Утверждение 2. Состояния s_1 и s_2 временного автомата S эквивалентны, если и только если в соответствующем конечном автомате A_S эквивалентными являются состояния (s_1, 0) и (s_2, 0).

Доказательство. Пусть состояния s_1 и s_2 временного автомата S эквивалентны. Предположим, что состояния (s_1, 0) и (s_2, 0) соответствующего конечного автомата не являются эквивалентными, т.е. для них существует различающая последовательность. Тогда в силу утверждения 1, найдётся различающая последовательность и для состояний s_1 и s_2, что противоречит утверждению, что состояния s_1 и s_2 эквивалентны.

Пусть теперь состояния (s_1, 0) и (s_2, 0) эквивалентны в построенном конечном автомате A_S, однако состояния s_1 и s_2 временного автомата не являются эквивалентными, т.е. для них существует различающая последовательность. Тогда в силу утверждения 1, найдётся различающая последовательность и для состояний (s_1, 0) и (s_2, 0), что противоречит утверждению, что состояния (s_1, 0) и (s_2, 0) эквивалентны.

Таким образом, вывод об эквивалентности состояний временного автомата может быть сделан на основе соответствующих состояний конечно автоматной абстракции.

4. Минимизация числа состояний

Процедура минимизации состояний для классических конечных автоматов хорошо известна [1] и основана на разбиении множества состояний конечного автомата по отношению эквивалентности. Мы предлагаем использовать аналогичный подход для сокращения числа состояний временного автомата. В соответствии с результатами предыдущего раздела, разбиение множества состояний по отношению эквивалентности для временного автомата может быть получено на основе соответствующего разбиения для конечно автоматной абстракции. Далее, мы предлагаем алгоритм построения приведённой по состояниям формы временного автомата.

Алгоритм 1 построения приведенной по состояниям формы полностью определённого детерминированного временного автомата

Вход. Полностью определённый детерминированный временной автомат S

Выход. Приведённая по состояниям форма автомата S

Шаг 1. По исходному временному автомату S строится соответствующая конечно автоматная абстракция A_S, для которой строится разбиение E_{FSM} множества состояний на эквивалентные состояния.

Шаг 2. Строится разбиение E состояний автомата S на эквивалентные состояния следующим образом: состояния s_1 и s_2 автомата S принадлежат одному блоку разбиения E, если и только если состояния (s_1, 0) и (s_2, 0) принадлежат одному блоку разбиения E_{FSM}.

Шаг 3. Строится приведенная по состояниям форма B автомата S. Входной и выходной алфавиты автомата B совпадают с таковыми для автомата S, состояния b_1, b_2, …, b_l соответствуют блокам B_1, B_2, …, B_l разбиения E. Множество переходов h_B автомата B формируем следующим образом. Выбираем из каждого блока B_i состояние s_i. Для состояний b_i и b_j существует кортеж $(b_i, i, o, b_j, g, d) \in h_B$, тогда и только тогда, когда существует состояние $s_j \in B_j$, такое что кортеж $(s_i, i, o, s_j, g, d) \in h_s$. Для функции таймаута $\Delta_B(b_i) = (b_j, T)$, тогда и только тогда, когда $\Delta_S(s_i) = (s_j, T)$, $s_j \in B_j$.

Следующее утверждение подтверждает корректность построения приведенной по состояниям формы временного автомата по описанному выше алгоритму.

Утверждение 3. Пусть B – автомат, построенный для автомата S по алгоритму 1. Автомат B является детерминированным полностью определенным автоматом, и состояние b_i автомата B эквивалентно состоянию s автомата S, если и только если $s \in B_i$.

Доказательство. Согласно утверждению 2, состояния s_1 и s_2 временного автомата S принадлежат одному блоку разбиения E, если и только если состояния $(s_1, 0)$ и $(s_2, 0)$ конечно автоматной абстракции A_S принадлежат одному блоку разбиения E_{FSM}. Таким образом, разбиение E, полученное на втором шаге алгоритма, есть разбиение множества состояний автомата S по отношению эквивалентности.

Автомат B является полностью определенным и детерминированным автоматом, поскольку автомат S обладает этими свойствами и для каждого состояния b_i множество переходов «повторяет» множество переходов для состояния $s \in B_i$.

Покажем, что состояния $s_i \in B_i$ и состояние b_i автомата B, соответствующее блоку B_i разбиения E, эквивалентны. Все переходы из состояния b_i дублируют переходы из некоторого состояния $s_i' \in B_i$, эквивалентного любому состоянию s_i из B_i (утверждение 2). В то же время, если $\Delta_S(s_i') = (s_k, T)$ и $\Delta_B(b_i) = (b_k, T)$, $T < \infty$, то $s_k \in B_k$ есть состояние, эквивалентное состоянию s_k', переходы которого «дублируются» в состоянии b_k. Аналогичные рассуждения можно провести и для возможных преемников по таймаутам состояний s_k и b_k. Таким образом, реакции на любой временной входной символ в состояниях s_i и b_i совпадают. Поскольку переходу $(s_i, i, o, s_i', g_i', d)$ исходного автомата соответствует переход $(b_i, i, o, b_i', g_i', d)$ приведённой по состояниям формы, где $s_i' \in B_i'$, то для состояний s_i' и b_i', и для всех их преемников по входным временным символам, можно провести аналогичные рассуждения. Таким образом, реакции автоматов B и S в состояниях b_i и s_i на любую входную последовательность совпадают. Покажем теперь, что состояния $s_j \in B_j$ и b_i, где $i \neq j$, не эквивалентны. Состояние $s_i \in B_i$ эквивалентно состоянию b_i согласно первой части доказательства. Состояния s_i и s_j не эквивалентны, так как находятся в различных блоках разбиения E. Таким образом, состояния s_j и b_i не эквивалентны.

Справедлива следующая теорема.

Теорема 1. Пусть S – полностью определённый детерминированный автомат с таймаутами и временными ограничениями. Полностью определённый детерминированный автомат B, построенный по алгоритму 1, является приведённой по состояниям формой автомата S.

В качестве примера рассмотрим автомат на рисунке 1. Приведенная по состояниям форма автомата S представлена на рисунке 2.

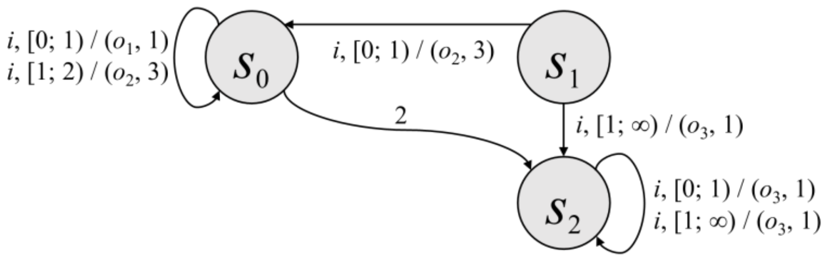


Рис. 2. Приведённая по состояниям форма временного автомата S
Fig. 2. State reduced form of TFSM S

Сложность алгоритма минимизации временного автомата с n состояниями, вообще говоря, определяется сложностью минимизации конечно автоматной абстракции, число состояний которой не превышает величины $2Bn$ и множество входных символов совпадает с таковым для временного автомата. Таким образом, сложность алгоритма минимизации полиномиальна относительно произведения наибольшей конечной границы для временных входных интервалов и числа состояний.

5. Минимизация временных аспектов

В отличие от классических автоматов, приведенная по состояниям форма не является минимальной формой временного автомата, т.к. не учитывает минимизацию временных параметров. В настоящем разделе мы вводим понятие минимальной формы временного автомата, которая является единственной с точностью до изоморфизма. Автомат с таймаутами и временными ограничениями S называется *приведенным по времени* (*time reduced*), если для любых двух кортежей (s, i, o, s', g_1, d), $(s, i, o, s', g_2, d) \in h_s$ справедливо, что g_1 и g_2 нельзя объединить в один интервал и для любого состояния s, такого что $\Delta_S(s) = (s', T)$, не существует состояния s'' и целого числа $T' < T$, таких что автомат S', полученный изменением функции таймаута состояния s автомата S на $\Delta_S(s) = (s'', T')$ эквивалентен исходному автомату S.

Если для переходов (s, i, o, s', g_1, d), $(s, i, o, s', g_2, d) \in h_s$ временного автомата S интервалы g_1 и g_2 могут быть объединены в один интервал, то эти два перехода можно заменить одним переходом вида $(s, i, o, s', g_1 \cup g_2, d)$. Поверка существования и поиск таймаута меньшей величины могут быть выполнены по конечно автоматной абстракции A_S. Для поиска минимального таймаута в состоянии s необходимо рассмотреть состояния абстракции вида (s, n), соответствующие поведению временного автомата в состоянии s в целочисленные моменты времени. Если для некоторого состояния (s, j) существует эквивалентное состояние вида $(s', 0)$, то переход по специальному входному символу $((s, (j - 1, j)), \mathtt{I}, \mathtt{I}, (s, j))$ может быть заменён на переход $((s, (j - 1, j)), \mathtt{I}, \mathtt{I}, (s', 0))$. Соответственно, таймаут в состоянии s автомата S, может быть заменён на $\Delta_S(s) = (s', j)$. Таким образом, для каждого состояния может быть выбран минимальный возможный таймаут, который не изменяет поведение временного автомата.

Далее, мы предлагаем алгоритм построения приведенной по времени формы временного автомата.

Алгоритм 2 построения приведенной по времени формы автомата с таймаутами и временными ограничениями

Вход. Детерминированный полностью определённый временной автомат S

Выход. Приведенная по времени форма автомата S

Шаг 1. Каждая пара переходов (s, i, o, s', g_1, d), (s, i, o, s', g_2, d), такая что g_1 и g_2 могут быть объединены в один интервал, заменяется на переход $(s, i, o, s', g_1 \cup g_2, d)$.

Шаг 2. По исходному автомату S строится соответствующий классический конечный автомат A_S, для которого строится разбиение по отношению эквивалентности.

Шаг 3. Для каждого состояния s автомата S, для которого $\Delta_S(s) = (s', T)$, среди состояний (s, j) автомата A_S, $j = 1, \ldots, T - 1$, определяется состояние с минимальным j, для которого существует эквивалентное состояние вида $(s'', 0)$. Если такое состояние s'' существует, то таймаут в состоянии s заменяется на $\Delta_S(s) = (s'', j)$. Если такого состояния нет, то таймаут в состоянии s не изменяется.

Шаг 4. Для каждого состояния s автомата S, для которого $\Delta_S(s) = (s', \infty)$, определяется состояние вида $(s'', 0)$ в автомате A_S, эквивалентное состоянию $(s, (j, \infty))$. Если такое состояние s'' существует, то таймаут в состоянии s заменяется на $\Delta_S(s) = (s'', j + 1)$. Если такого состояния нет, то таймаут в состоянии s не изменяется.

Шаг 5. Для каждого состояния s, где $\Delta_S(s) = (s', T)$ и $T < \infty$, удаляются все переходы (s, i, o, s', g, d), такие что $g \cap [0, T) = \varnothing$. Если для перехода $(s, i, o, s', \langle a, b\rangle, d)$ справедливо, что $\langle a, b\rangle \cap [0, T) \neq \varnothing$ и неверно, что $\langle a, b\rangle \subseteq [0, T)$, то переход $(s, i, o, s', \langle a, b\rangle, d)$ заменяется на переход $(s, i, o, s', \langle a, T), d)$.

Имеет место следующее утверждение.

Утверждение 4. Пусть P – временной автомат, построенный для приведенного по состояниям автомата S по алгоритму 2. Автомат P эквивалентен автомату S и является приведенным по времени автоматом.

Доказательство. Покажем, что автоматы S и P эквивалентны. Объединение переходов (s, i, o, s', g_1, d), (s, i, o, s', g_2, d) в переход $(s, i, o, s', g_1 \cup g_2, d)$ (Шаг 1) не изменяют поведение автомата. Рассмотрим процедуру замены таймаута на третьем шаге алгоритма. Пусть для состояния p временного автомата P найдено состояние временного автомата p'', такое, что существует состояние (p, j) из A_P, эквивалентное состоянию $(p'', 0)$ автомата A_P. Здесь $\Delta_P(p) = (p', T)$ и $j < T$. Последнее означает, в силу утверждения 2, что реакция автомата P в состоянии p'' на любую входную последовательность совпадает с таковой для автомата в состоянии p в момент времени j. Таким образом, при замене функции таймаута $\Delta_P(p) = (p', T)$ в состоянии p на $\Delta_P(p) = (p'', j)$ поведение автомата P в состоянии p не меняется. Аналогично, замена таймаута на четвёртом шаге алгоритма не изменяет поведение автомата. На пятом шаге алгоритма удаляются лишь переходы (p, i, o, p', g, d), временные интервалы которых не пересекаются с интервалом $[0, T)$, где $\Delta_P(p) = (p', T)$, соответственно, такие переходы не могут быть выполнены автоматом. Переходы, временные интервалы которых пересекаются с $[0, T)$, но выходят за границу таймаута, ограничиваются сверху величиной таймаута, что также не изменяет поведение автомата.

Покажем теперь, что автомат P является приведенным по времени автоматом. Допустим, что для некоторого состояния p_0 автомата P таймаут $\Delta_P(p_0) = (p_1, T_1)$ может быть заменён на таймаут $\Delta_P(p_0) = (p_2, T_2)$, где $T_1 > T_2$, таким образом, что поведение автомата в состоянии p_0 не измениться. Последнее означает, что в конечно автоматной абстракции A_P существует либо пара эквивалентных состояний (p_0, T_2) и $(p_2, 0)$, либо $(p_0, (T_2 - 1, \infty))$ и $(p_2, 0)$. Существование эквивалентных состояний (p_0, T_2) и $(p_2, 0)$ невозможно по третьему шагу алгоритма. Существование эквивалентных состояний $(p_0, (T_2 - 1, \infty))$ и $(p_2, 0)$ отражает тот факт, что реакция на любую входную временную последовательность автомата P в состоянии p_0 в момент времени $T > (T_2 - 1)$ совпадает с таковой для состояния $(p_2, 0)$. В таком случае, состояния (p_0, T_2) не существует по построению абстракции, однако, по шагу 4 алгоритма, таймаут был бы заменён на $\Delta_P(p_0) = (p_2, T_2)$. Таким образом, существование эквивалентных состояний $(p_0, (T_2 - 1, \infty))$ и $(p_2, 0)$ невозможно.

В то же время, любые два перехода вида (s, i, o, s', g_1, d), (s, i, o, s', g_2, d), которые возможно объединить в один переход $(s, i, o, s', g_1 \cup g_2, d)$, объединяются на шаге 1 алгоритма 2.

На основе полученных результатов, можно установить необходимое и достаточное условие эквивалентности временных автоматов.

Теорема 2. Детерминированные полностью определенные приведённые по состояниям и времени временные автоматы S и P эквивалентны, если и только если эти автоматы изоморфны.

Необходимость. Пусть S и P – детерминированные полностью определённые приведённые по состояниям и времени автоматы с временными ограничениями и таймаутами, которые являются эквивалентными. Покажем, что автоматы S и P изоморфны. Рассмотрим отображение H: $S \rightarrow P$, при котором $H(s)$ есть состояние автомата P, эквивалентное состоянию s. Отображение H является взаимно однозначным в силу приведённости по состояниям автоматов S и P. Покажем, что для любой пары состояний s и $p = H(s)$, такой что $\Delta_S(s) = (s', T_s)$, $\Delta_S(p) = (p', T_p)$, справедливо, что $T_s = T_p$ и $p' = H(s')$. Допустим, что $T_p < T_s$; тогда в силу эквивалентности S и P, существует состояние s'', эквивалентное состоянию p'. А поскольку состояния s и p также являются эквивалентными, то таймаут в состоянии s может быть заменён на $\Delta_S(s) = (s'', T_s')$, где $T_s' = T_p < T_s$. Последнее невозможно, поскольку S является приведенным по времени автоматом. Пусть теперь $T_p > T_s$, тогда в силу эквивалентности S и P, существует состояние p'', эквивалентное состоянию s'. А поскольку состояния s и p также являются эквивалентными, то таймаут в состоянии p может быть заменён на $\Delta_P(p) = (p'', T_p')$, где $T_s = T_p' < T_p$. Последнее невозможно, поскольку P есть приведенный по времени автомат. В силу равенства таймаутов $T_p = T_s$ и эквивалентности состояний s и p, справедливо, что функции $time(s, T_s)$ и $time(p, T_p)$ совпадают, и, соответственно, $p' = H(s')$. Аналогично [11], ввиду эквивалентности S и P, справедливо, что для любого перехода (s, i, o, s', g, d) существует переход $(H(s), i, o, H(s'), g, d)$. Таким образом, автоматы S и P изоморфны.

Достаточность. Поскольку изоморфные автоматы совпадают с точностью до переименования состояний, то изоморфные автоматы эквивалентны.

Для приведённой по состояниям формы автомата S (рисунок 2) приведенная по времени форма показана на рисунке 3.

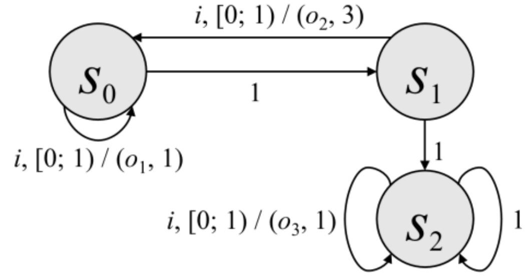


Рис. 3. Приведённая по времени и состояниям форма автомата S
Fig. 3. Time and state reduced form of reduced form of S

Как видно из рисунка 3, таймаут $\Delta_S(s_0) = (s_2, 2)$ был заменён на таймаут $\Delta_S(s_0) = (s_1, 1)$, а переход $(s_0, i, o_2, s_0, [1, 2), 3)$ удалён. В то же время, два перехода из состояния s_2 были объединены, а таймауты $\Delta_S(s_1) = (s_1, \infty)$ и $\Delta_S(s_2) = (s_2, \infty)$ заменены на $\Delta_S(s_1) = (s_2, 1)$ и $\Delta_S(s_2) = (s_2, 1)$ соответственно.

Таким образом, для временных автоматов необходимо ввести определение минимальной формы, учитывающей не только минимизацию состояний, но и минимизацию временных аспектов. *Минимальной формой* полностью определённого детерминированного автомата с таймаутами и временными ограничениями S называется приведённый по состояниям и времени автомат, эквивалентный S. По определению минимальной формы, справедлива следующая теорема.

Теорема 3. Минимальная форма автомата с таймаутами и временными ограничениями единственна с точностью до изоморфизма.

7. Заключение

В настоящей работе предложен метод построения минимальной формы автомата с таймаутами и временными ограничениями. Предложенный подход основан на построении разбиения множества состояний временного автомата на классы эквивалентности, которое строится на основе соответствующего разбиения для конечно автоматной абстракции. В отличие от классических конечных автоматов, предложенный подход минимизирует не только число состояний, но и временные аспекты. Мы также показываем, что минимальная форма временного автомата единственна с точностью до изоморфизма. Тем не менее, можно показать, что свойство минимальности временного автомата не сохраняется для его конечно автоматной абстракции. В конечно автоматной абстракции минимального временного автомата возможно наличие эквивалентных состояний. Таким образом, представляет интерес исследование свойств конечно автоматной абстракции временных автоматов, для которых свойство минимальности будет инвариантом.

Отметим также, что единственная минимальная форма для автоматов только с таймаутами и только с временными ограничениями может быть построена по алгоритму 2 с необходимыми сокращениями.

Представляет интерес изучение минимальной формы для инициальных автоматов с таймаутами, т.е. автоматов, в которых выделено начальное состояние. В этом случае в исходном автомате возможно наличие состояний, которые достижимы из начального состояния только по таймаутам и соответственно, не обязательно имеют эквивалентное состояние в минимальной форме. В качестве дальнейших исследований мы также предполагаем провести эксперименты по минимизации временных автоматов, описывающих поведение реальных управляющих систем.

Благодарности

Работа выполнена при поддержке гранта РНФ No. 16-49-03012.

Список литературы

[1]. Gill A. Introduction to the Theory of Finite-State Machines. 1962, 207 p.

[2]. Lee, D., Yannakakis, M. (1996) Principles and methods of testing finite state machines-a survey. Proceedings of the IEEE. 1996, 84(8), pp. 1090-1123.

[3]. Murphy T.E., Geng X.-J., Hammer J.. On the control of asynchronous machines with races. IEEE Transactions on Automatic Control, 2003, 48(6), pp. 1073-1081.

[4]. Kumar R., Garg V.K. Modeling and control of logical discrete event systems. Kluwer Academic Publishers, 1995, 143 p.

[5]. Cassandras C. C., Lafortune S.. Introduction to discrete event systems. Kluwer Academic Publishers, 1999, 822 p.

[6]. Dorofeeva R., El-Fakih K., Maag S., Cavalli A., Yevtushenko N. FSM-based conformance testing methods: A survey an-notated with experimental evaluation. Information and Software Technology, 2010, 52, pp. 1286-1297.

[7]. Zhigulin M., Yevtushenko N, Maag S., Cavalli A. FSM-Based Test Derivation Strategies for Systems with Time-Outs. Proc. of the 11th International Conference on Quality Software, QSIC 2011, IEEE, 2011. pp. 141-149.

[8]. El-Fakih K., Yevtushenko N., Fouchal H. Testing Timed Finite State Machines with Guaranteed Fault Coverage. TestCom/FATES, 2009, pp. 66-80.

[9]. Merayo M., Nunez M., Rodriguez I., Formal testing from timed finite state machines, Comput. Netw, 2008, 52 (2), 432-460.

[10]. Bresolin D., El-Fakih K., Villa T., Yevtushenko N. Deterministic Timed Finite State Machines: Equivalence Checking and Expressive Power. Intern Conf. GANDALF, 2014, pp. 203-216.

[11]. Твардовский А. С., Евтушенко Н. В. К минимизации автоматов с временными ограничениями. Вестник ТГУ. Управление, вычислительная техника и информатика, № 4 (29), 2014 г., стр. 77-83.

[12]. Твардовский А. С. К минимизации автоматов с таймаутами. Труды ИСП РАН, т. 26, вып. 6, 2014 г., стр. 77-84. DOI: 10.15514/ISPRAS-2014-26(6)-7

Minimizing Finite State Machines with time guards and timeouts

A.S. Tvardovskii <tvardal@mail.ru>
N.V. Yevtushenko <nyevtush@gmail.com>
M.L. Gromov <maxim.leo.gromov@gmail.com>
National Research Tomsk State University
Lenin st. 36, Tomsk, 634050, Russia

Abstract. Finite State Machines (FSMs) are widely used for analysis and synthesis of components of control systems. In order to take into account time aspects, timed FSMs are considered. As the complexity of many problems of analysis and synthesis of digital and hybrid systems including high-quality test derivation significantly depends on the size of the system (component) specification, in this paper, we address the problem of minimizing a FSM with timed guards and input and output timeouts (TFSM). The behavior of a TFSM can be described using a corresponding FSM abstraction and a proposed approach for minimizing a TFSM is based on such abstraction. We minimize not only the number of states as it is done for classical FSMs but also the number of timed guards and timeout duration. We show that for a complete deterministic TFSM there exists the unique minimal (canonical) form, i.e., a unique time and state reduced TFSM that has the same behavior as the given TFSM; for example, this minimal form can be used when deriving tests for checking whether an implementation under test satisfies functional and nonfunctional requirements. A proposed approach for minimizing timed machines can be applied to particular cases of TFSM, i.e. for FSM with timeouts and FSM with timed guards.

Keywords: Timed Finite State Machines; reduced form; minimal form.

DOI: 10.15514/ISPRAS-2017-29(4)-8

For citation: Tvardovskii A.S., Yevtushenko N.V., Gromov M.L. Minimizing Finite State Machines with time guards and timeouts. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017 г., pp. 139-154 (in Russian). DOI: 10.15514/ISPRAS-2017-29(4)-8

References

[1]. Gill A. Introduction to the Theory of Finite-State Machines. 1962, 207 p.

[2]. Lee, D., Yannakakis, M. (1996) Principles and methods of testing finite state machines-a survey. Proceedings of the IEEE. 1996, 84(8), pp. 1090-1123.

[3]. Murphy T.E., Geng X.-J., Hammer J.. On the control of asynchronous machines with races. IEEE Transactions on Automatic Control, 2003, 48(6), pp. 1073-1081.

[4]. Kumar R., Garg V.K. Modeling and control of logical discrete event systems. Kluwer Academic Publishers, 1995, 143 p.

[5]. Cassandras C. C., Lafortune S.. Introduction to discrete event systems. Kluwer Academic Publishers, 1999, 822 p.

[6]. Dorofeeva R., El-Fakih K., Maag S., Cavalli A., Yevtushenko N. FSM-based conformance testing methods: A survey an-notated with experimental evaluation // Information and Software Technology, 2010, 52, pp. 1286-1297.

[7]. Zhigulin M., Yevtushenko N, Maag S., Cavalli A. FSM-Based Test Derivation Strategies for Systems with Time-Outs. Proc. of the 11th International Conference on Quality Software, QSIC 2011, IEEE, 2011. pp. 141-149.

[8]. El-Fakih K., Yevtushenko N., Fouchal H. Testing Timed Finite State Machines with Guaranteed Fault Coverage. TestCom/FATES, 2009, pp. 66-80.

[9]. Merayo M., Nunez M., Rodriguez I., Formal testing from timed finite state machines, Comput. Netw, 2008, 52 (2), 432-460.

[10]. Bresolin D., El-Fakih K., Villa T., Yevtushenko N. Deterministic Timed Finite State Machines: Equivalence Checking and Expressive Power. Intern Conf. GANDALF, 2014, pp. 203-216.

[11]. Tvardovskii A., Yevtushenko N. Minimizing timed Finite State Machines. Tomsk State University Journal of Control and Computer Science, 2014, № 4 (29), pp. 77-83 (in Russian).

[12]. Tvardovskii A. On the minimization of timed Finite State Machines. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 6, 2014, pp. 77-84 (in Russian). DOI: 10.15514/ISPRAS-2014-26(6)-7

Mining Hybrid UML Models from Event Logs of SOA Systems

K.V. Davydova <kvdavydova@edu.hse.ru>
S.A. Shershakov <sshershakov@hse.ru>
National Research University Higher School of Economics,
PAIS Lab at the Faculty of Computer Science,
20 Myasnitskaya st., Moscow, 101000, Russia

Abstract. In the paper we consider a method for mining so-called “hybrid” UML models, that refers to software process mining. Models are built from execution traces of information systems with *service-oriented architecture* (SOA), given in the form of event logs. While common reverse engineering techniques usually require the source code, which is often unavailable, our approach deals with event logs which are produced by a lot of information systems, and some heuristic parameters. Since an individual type of UML diagrams shows only one perspective of a system’s model, we propose to mine a combination of various types of UML diagrams (namely, *sequence* and *activity*), which are considered together with *communication* diagrams. This allows us to increase the expressive power of the individual diagram. Each type of diagram correlates with one of three levels of abstraction (workflow, interaction and operation), which are commonly used while considering web-service interaction. The proposed algorithm consists of four tasks. They include splitting an event log into several parts and building UML sequence, activity and communication diagrams. We also propose to encapsulate some insignificant or low-level implementation details (such as internal service operations) into activity diagrams and connect them with a more general sequence diagram by using *interaction use* semantics. To cope with a problem of immense size of synthesized UML sequence diagrams, we propose an abstraction technique based on regular expressions. The approach is evaluated by using a developed software tool as a Windows-application in C#. It produces UML models in the form of XML-files. The latter are compatible with well-known Sparx Enterprise Architect and can be further visualized and utilized by that tool.

Keywords: event log, process mining, hybrid UML model, UML sequence diagram, UML activity diagram, reverse engineering.

DOI: 10.15514/ISPRAS-2017-29(4)-10

For citation: Davydova K.V., Shershakov S.A. Mining Hybrid UML Models from Event Logs of SOA Systems. *Trudy ISP RAN/Proc. ISP RAS,* vol. 29, issue 4, 2017, pp. 155-174. DOI: 10.15514/ISPRAS-2017-29(4)-10

1. Introduction

Nowadays we use information systems everywhere. They are used not only at home to increase the comfort of our life but also to support business processes. The complexity of the systems is growing together with the complexity of processes and tasks. Moreover, a lot of systems interact with each other. There is an increasing chance of error as the complexity of the system increases. If the system finds these errors, they are written into so-called event logs together with other information about system execution. The logs store a lot of information during the work of the system. On the one hand, manual processing of the logs is almost impossible because of their size and lack of structure. On the other hand, the event logs are an inestimable source of knowledge about real-life system behavior. Tools, which help to obtain this knowledge in suitable form for analytics are extremely useful.

Different approaches, such as modeling, development within the standardized life cycle, testing, quality assurance (QA), verification, etc., are applied to improve the system quality and error correction. Using combinations of these instruments (for example, testing and verification, modeling and reverse engineering with continuous delivery) gives good results. New tools, modeling tools in particular, help to make the process more convenient and more effective.

Models are built on different life cycle stages. In the classic approach, an architect models an information system based on the customer‘s requirements. However, the implemented system often differs from previously developed models because the system is developed faster than its models. Developers may sometimes make mistakes and may need to spend additional time on critical situations and deadlines. This means that the design and implementation of some components is not completed properly.

When there is no complete model of a system, reverse engineering techniques can be applied to extract the necessary information from the system and build an appropriate model. It allows us to obtain models of a real-life system automatically or semi-automatically. These models correspond to a developed system rather than to an initial plan and initial models. Such models aim both to understand a structure/behavior of a real system and to eliminate any inadequacy of a real model as compared to the initial model. This also makes it easier to fix errors in the system. There are a number of approaches and tools aimed for this purpose. Most of them require the source code of a system to perform analysis. It is not always possible because of different reasons: the source code may not be available to analysts; it is impossible to get the last copy of code or it can be lost. Moreover, different work groups can develop different system components which complicates centralized collection of source code.

Unlike existing reverse engineering approaches that use source code, we propose an approach that works with system execution traces which can be extracted from event logs. Our approach can be considered as a particular implementation of Process Mining [1], a discipline aimed to discover, analyze and improve business processes and their models. Our approach also includes features that are relevant to software engineering. Hence, we refer to it as *software process mining* [2].

Process mining usually uses process models such as Petri nets, BPMN, Fuzzy maps, etc. which are produced by applying different algorithms such as α-algorithm [1], [3], [4], NLP-algorithm [5] or fuzzy miner [6] respectively. However, these models are not perfectly suitable for software developers. In the software engineering area, more specific approaches such as the Unified Modeling Language (UML) [7] are more common. The most common approaches deal with *static class diagrams*, *statecharts*, *sequence* and *activity* diagrams considering them as more descriptive than other. According to UML 2.5, there are two groups of diagrams: *structural* and *behavioral*. In this work we primarily focus on the behavioral group, in particular, on *sequence*, *activity* and *communication* diagrams.

Modern approaches to the development of information systems make out small reusable well-defined pieces of code, which are commonly refered to as *services*. Systems, using services as a main component, are based on *service-oriented architecture* (SOA) [8]. Services from heterogeneous SOA-systems are developed using different languages, environments and tools, but they work in a single *information space*. Mining unified models of those systems is a challenge and has some difficulties. For example, none of the popular reverse engineering tools works with all languages used for web-service development [9]. As almost all systems produce event logs which contain information about interesting system components, it is possible to build models including all of these components. It simplifies the process of reverse engineering and allows us to expand its application area.

In the paper, we consider event logs written by SOA-systems. Our goal is to expand the applicability of UML-based models for SOA-systems by developing new approaches and tools for mining such models from event logs. UML standard describes different types of models which suit different modeling aspects of an information system. Nevertheless, there are situations when analysts would like to use expressive opportunities of several diagram types. UML 2.5 does not describe such diagrams, and it does not forbid them either. In our paper, we propose a new approach to UML-modeling, which includes mining a so-called hybrid diagram that comprises elements of UML *sequence* and UML *activity diagrams*.

To illustrate the proposed approach, consider the following example.

1.1. Motivating example

We consider an event log (Table I) produced by an online banking information system with service-oriented architecture. The log contains a number of traces corresponding to individual instances of a business process maintained by the information system. Our goal is to obtain a UML model that represents some behavioral aspects of the system from different perspectives [9].

Each row of Table I represents a single event. Columns represent attributes of the log. Events are grouped in cases (by CaseID attribute); then, cases are represented in the log by traces. Events are ordered by Timestamp attribute. Different components of SOA are represented by other attributes such as Domain, Service/Process and Operation. Domains contain services and processes while the latter consist of operations [10].

Table 1. Log fragment L1. Banking SOA-system

CaseID	Domain	Service/Process	Operation	Action	Payload	Timestamp
23	Account	Operations	GetLastOperations	REQ	user=a, today=23.07.2015, client=Maria, manager=Julia	17:32:15 135
23	Account	CardInfo	GetCardID	REQ	user=a, num=0	17:32:15 250
23	Account	CardInfo	GetCardInfo	REQ	num=0	17:32:15 260
23	Account	CardInfo	GetCardInfo	RES	date=07/16, name=MARIA GRISHINA, id=15674839	17:32:15 267
23	Account	CardInfo	GetCardID	RES	res=15674839	17:32:15 297
23	Card	Operations	GetOperations	REQ	days=30	17:32:15 378
23	Utils	Calendar	GetDate	REQ	days=30	17:32:15 409
23	Utils	Calendar	GetDate	RES	res=23.06.2015	17:32:15 478
23	Card	Operations	GetOperations	RES	res={BP Billing Transfer}	17:32:15 513
23	Card	OperationData	GetPlaceAndDate	REQ	op=BP Billing Transfer	17:32:15 559
23	Card	OperationData	GetPlace	REQ	op=BP Billing Transfer	17:32:15 563
23	Card	OperationData	GetPlace	RES	res=RUS SBERBANK ONLAIN PLATEZH	17:32:15 571
23	Card	OperationData	GetDate	REQ	op=BP Billing Transfer	17:32:15 575
23	Card	OperationData	GetDate	RES	res=20.07.2015	17:32:15 589
23	Card	OperationData	GetPlaceAndDate	RES	res=RUS SBERBANK ONLAIN PLATEZH, date=20.07.2015	17:32:15 601
23	Account	Operations	GetLastOperations	RES	res=succ	17:32:15 822
25	Account	Operations	GetLastOperations	REQ	user=a, today=23.07.2015, client= Maxim, manager=Julia	17:40:18 345
25	Account	CardInfo	GetCardID	REQ	user=a	17:40:18 408
25	Account	CardInfo	GetCard	REQ	num=0	17:40:18 422
25	Account	CardInfo	GetCard	RES	res=no cards	17:40:18 434
25	Account	CardInfo	GetCardID	RES	res=error	17:40:18 489
25	Account	Operations	GetLastOperations	RES	res=no bounded cards	17:40:18 523

By applying a method [9] to the example log, we obtain a UML sequence diagram as depicted in Figure 1 representing the overall process. The diagram contains all possible details (excluding operation parameters) of the behavior of the system as it is represented in the event log. Along with regular messages which connect two different lifelines (depicted as vertical dash lines), the diagram also contains a number of self-calls represented as labeled loop arrows, e.g. `GetCardInfo`, `GetCard`. These self-calls are not important for studying the model from a more abstract perspective. In contrast, they are important when modeling the process of the individual service or another SOA component.

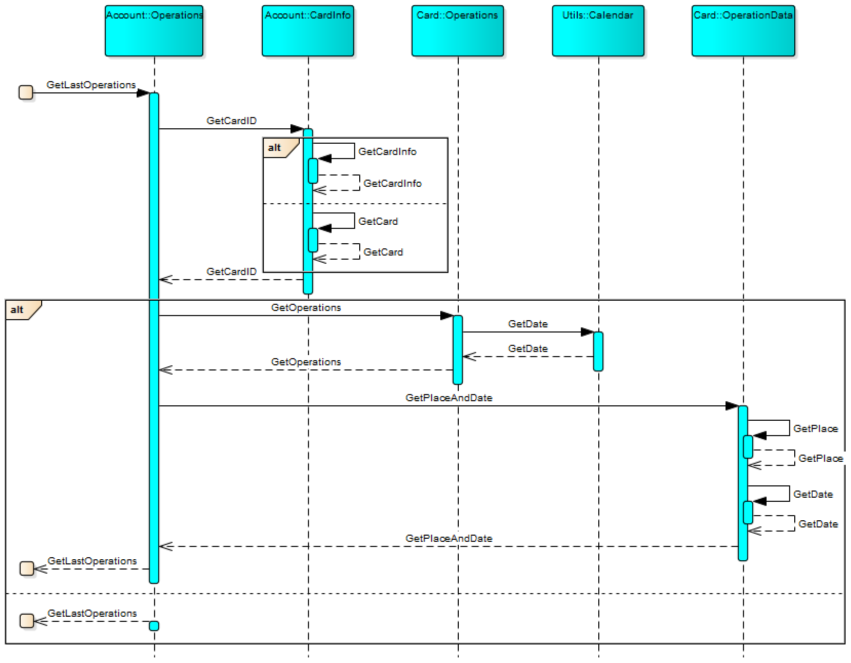


Fig. 1. Usual UML sequence diagram mined from event log L1.

Thus, we propose to hide these calls on the general model with giving a reference to another diagram. Note, that the hidden calls are restricted by one lifeline only. So, using UML sequence diagram here loses its meaning, since only one agent is involved. Therefore, it is convenient to model such behavior by using *UML activity diagrams*, another type of UML diagram. Figures 2, 3 and 4 illustrate this idea and represent a *hybrid UML diagram* combining the best features of two different model types.

A distinctive feature of SOA, which is considered, is that processes call other processes and services while services do not call other participants. To demonstrate this feature, it is important to show the interaction between one selected service and its direct services-neighbors which the service communicates with. A UML communication diagram suits this purpose. Example diagrams for `Card::Operations` and `Card::OperationData` processes from example event log are depicted in Figures 5 and 6 respectively. We can see that these processes are called by other processes and call both different services and themselves.

We developed a tool that builds hybrid diagrams of UML sequence and activity diagrams automatically. Moreover, the tool is able to build a UML communication diagram for a selected SOA component.

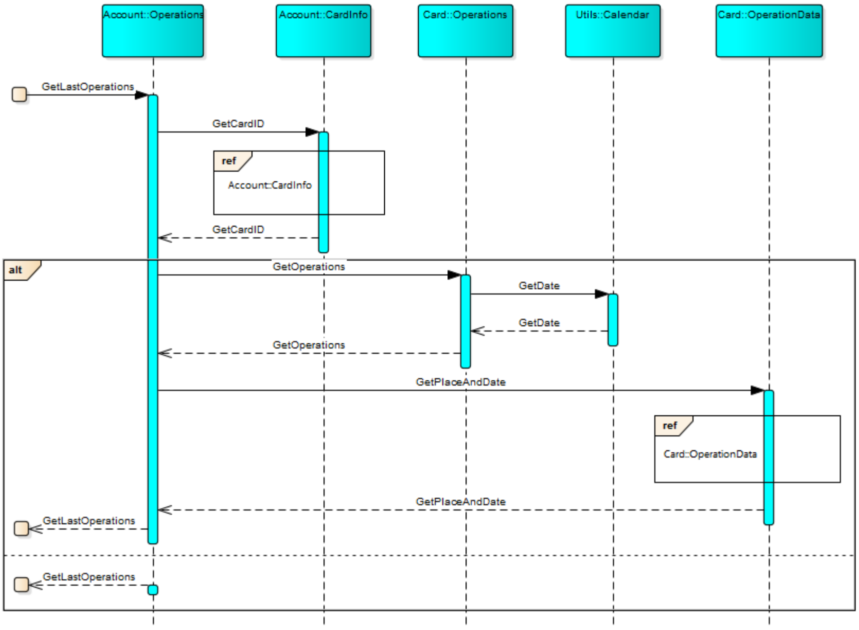


Fig. 2. UML sequence diagram with hidden self calls. High-level diagram of a hybrid UML diagram.

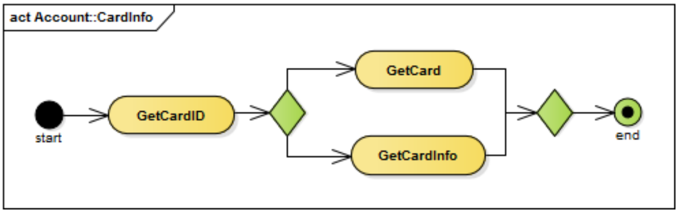


Fig. 3. UML activity diagram with an activity inside Account::CardInfo service.

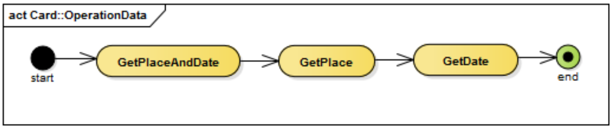


Fig. 4. UML activity diagram with an activity inside Card::OperationData service.

1.2. Related work

Reverse engineering of behavioral UML diagrams is not a new area. There are a number of works [11], [12], [13], [14], about building the UML diagrams based on static source code analysis. Besides, there are some CASE tools [15], [16], [17], [18], which can be used for reverse engineering of sequence and activity UML diagrams. There is also a plug-in [19] for NetBeans development environment that is able to build different types of behavioral models from Java source code.

However, all of the methods and tools mentioned above use static program analysis (getting models from source code without execution) for their work. As it was considered earlier, source code and all of its versions are not always available for analysis. Hence, these tools and methods are useless in this case. Furthermore, none of these tools is able to infer models from the code written in most popular languages used for developing SOA information systems. Moreover, SOA architectures are often developed with various programming languages. For example, some modules can be written in C#, whereas others can be developed in Java; they can interact with LAMP service, so a single CASE tool cannot produce models for that system. Mining diagrams from event logs solves this problem.

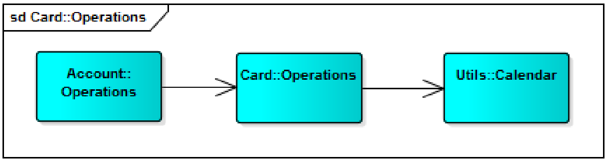


Fig. 5. UML communication diagram for Card::Operations service.

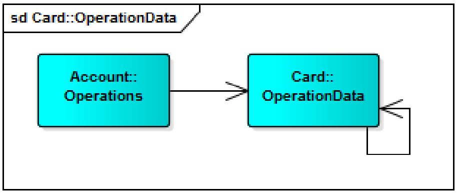


Fig. 6. UML communication diagram for Card::OperationData service.

In [20], [21], [22], approaches to building models based on execution traces are proposed. One related work [20] analyzes a single trace using meta-models of an event log trace and a UML sequence diagram (UML SD). The trace includes information not only about invocation of methods but also about loops and conditions, which makes easier recognition of fragments such as iteration, alternatives and options. However, logs of information systems do not usually include this information, so it is necessary to modify the source code to apply this approach.

There is a description of the mining UML sequence diagrams method based on several execution traces in [22]. The authors propose to use a labeled transition system (LTS) as an intermediate model to present one trace and an algorithm to merge LTSs built by several traces. After that, the LTS is transformed into a UML sequence diagram. Moreover, LTS can be used to build a Petri net that can then be converted into a UML activity diagram [23]. This conversion possibility can be used to apply different process mining algorithms for receiving a UML activity diagram. The approach to mining hierarchical UML sequence diagrams is proposed in [9] (see Section III-D).

In [24], the authors describe a framework which allows not only behavioral but also static UML diagrams to be built. Their framework generates execution traces by itself from Java source code. After that, the framework is able to build UML activity diagrams from traces, but it requires source code for its work.

Process mining proposes to use three abstraction levels for mining models for web services interaction [25]: workflow, interaction and operation. At the operation level, only one service is considered in order to look at its internal behavior and functionality. At the interaction level, they consider not only one selected service but also its direct callers and callees. Finally, the overall services interaction is covered at the workflow level. We apply all of these levels to service-oriented architecture in the paper.

Furthermore, research on service mining was described in [26]. The author builds different Petri nets for different services (considered at the operation level) and then combines them by places. Thus, he builds a generalized model which refers to the workflow level.

The rest of the paper is organized as follows. Section II gives definitions. Section III introduces our approach to mining hybrid UML models. Section IV contains a description of tool implementation. Section V concludes the paper and gives directions for further research.

2. Preliminaries

$\mathcal{P}(X)$ is the powerset over some set X; Λ is a set of all possible string labels.

Definition 1. (Event log) Let $e = (a_1, a_2, \dots, a_n)$ be an event, where a_i is an i-th attribute and n is a number of them. E is a set of events. $\sigma =< e_1, e_2, \dots, e_k >$ is an event trace where $e_1, e_2, \dots, e_k$ is an ordered set of events. $Log = \mathcal{P}(E)$ is an event log which is a powerset of traces.

Definition 2. (UML Sequence Diagram) A UML sequence diagram is a tuple $U_{SD} = (L, T, A, P, M, Ref, F)$, where:

- T is a set of moments of discrete time, which determine a partial order over diagram components.
- L is a set of named lifelines. $L = \{l = (\lambda, t) | \lambda \in \Lambda, t \in T\}$
- A is a set of activations mapped onto lifelines. $a \in A{:}\ a = (l, t_b, t_e)$, where $l \in L, t_b, t_e \in T, t_b < t_e$
- $P \subset \Lambda$ is a set of message parameters.
- *Ref* is a set of interaction use (ref fragments) which group lifelines and hide them interaction. $ref \in Ref : ref = (L', \lambda)$, where $L' \subset L, \lambda \in \Lambda$
- M is a set of messages. $m \in M{:}\ m = (a_1, t, \lambda, a_2, type)$, where $a_1, a_2 \in A \cup Ref, t \in T, \lambda \in P, type \in \{call, return\}$. $a_1 = (l_1, t_{11}, t_{12}), a_2 = (l_2, l_{21}, l_{22}){:}\ t_{11} \leq t_{21}, t_{11} < t_{12}, t_{21} < t_{22}$
- F is a set of combined fragments of the diagram. $F = \{(frag, M') | M' \subseteq M, frag \in \{alt, loop, opt, par\}\}$

Figure 1 represents an example of UML sequence diagram. A *lifeline* is represented as a vertical dashed line with its name at the top. An *activation* is represented as a rectangle on a lifeline, which takes and emits messages (represented as arrows). Message can be *call* and *return* and they contain text parameters. Messages inside one fragment are ordered by time. *Fragments* contain a number of messages and can contain other combined fragments. They are able to show alternatives, loops, parallelisms and other control structures. Another type of fragment, *ref* fragments, refer to other diagrams. Such diagrams can be both UML sequence diagrams and UML activity ones.

Definition 3. (UML Activity Diagram) A UML activity diagram is a tuple $U_{AD} = (N, E, NT)$, where:

- NT is a set of node types. $NT = \{control, object, executable\}$
- N is a set of nodes. $n \in N{:}\ n = (\lambda, type)$, where $\lambda \in \Lambda, type \in NT$
- E is a set of edges. $e \in E{:}\ e = (n_1, n_2)$, where $n_1, n_2 \in N$

Figure 3 represents an example of a UML activity diagram for `Account::CardInfo` service. Different node types have different meanings. Control nodes represent different behavioral elements such as start, fork and decision. Object nodes represent data (input and output) of an action. Executable nodes represent steps (actions) of the modeling activity. There are three named executable nodes and four control nodes (start, end, decision and merge) in Figure 3. Different control nodes can impose limitations. For instance, start nodes cannot have incoming edges, end nodes cannot have outgoing edges, decision and fork nodes can have only one incoming edge but several outgoing ones; the opposite is true for merge and join.

$\mathfrak{U}_{AD}$ is a set of all possible UML activity diagrams U_{AD}.

Definition 4. (Hybrid UML Diagram) A hybrid UML diagram is a tuple $U_{HD} = (U_{SD}, AD, f)$, where:

- $U_{SD} = (L, T, A, P, M, Ref, F)$ is a UML sequence diagram.

- $AD \subset \mathfrak{U}_{AD}$
- $f: Ref \to AD$ is a function which maps ref fragments from a UML sequence diagram onto corresponding activity diagram.

Figures 2, 3 and 4 illustrate an example of a hybrid UML diagram. Figure 2 is a UML sequence diagram and represents a high-level diagram. It refers to UML activity diagrams (Figures 3 and 4) using *ref* fragments.

Definition 5. (UML Communication Diagram) A UML communication diagram is a tuple $U_{CD} = (L_{CD}, M_{CD})$, where:

- $L_{CD} \subset \Lambda$ is a set of named lifelines which represent interaction participants.
- M_{CD} is a set of messages. $m_{CD} \in M_{CD}: m_{CD} = (l_1, l_2, \lambda)$, where $l_1, l_2 \in L_{CD}, \lambda \in \Lambda$.

Figures 5 and 6 provide examples of UML communication diagrams for two different services.

$\mathfrak{U}_{CD}$ is a set of all possible UML communication diagrams U_{CD}.

Definition 6. (Hybrid UML Model) A hybrid UML model is a tuple $U_{CD} = (U_{HD}, \mathrm{CD})$, where:

- U_{HD} is a hybrid UML diagram.
- $\mathrm{CD} \subset \mathfrak{U}_{CD}$.

Figures 2, 3, 4, 5 and 6 represent a hybrid UML model built for example event log L1.

3. Mining Hybrid UML Models

The authors in [25] propose definitions of three levels of abstraction: *operation*, *interaction* and *workflow*. The levels are used for consideration of web service interaction. It motivated us to use different types of UML diagrams which demonstrate features of these levels. In the following sections, we consider which UML diagrams suit each abstraction level and why.

3.1. Operation and workflow abstraction levels

Operation level of abstraction shows what is happening inside one isolated service. Activities outside the service are not considered at the operation level; the only process participants are services. Using a UML sequence diagram leads to a large number of self-calls and *“snowball models”*. It makes the diagram less readable and less understandable. A UML activity diagram suits this purpose since it allows us to demonstrate the complex relationships between operations inside a single participant. Figure 3 shows an example of a UML activity diagram for service `Card::OperationData`.

A business process, provided by services, is represented at a *workflow* abstraction level. There are a lot of participants, so it is useful to use a UML sequence diagram for this level. The diagram is suitable to present not only a sequence of business process actions but also participants of this process and their interaction. An example for event log L1 is depicted in Figure 1.

To bind different abstraction levels, it is necessary to connect them. Our proposal is to use *hybrid UML diagrams* to represent and connect *operation* and *workflow* abstraction levels together. A UML sequence diagram is used to represent a business process at a workflow abstraction level. The diagram contains special objects, *ref* fragments, which make a connection to corresponding UML activity diagram. Every such activity diagram models the behavior of a single service. An example of considered hybrid diagram is presented in Figures 2, 3 and 4.

```
Input  : an event log $Log$;
an attribute name with REQ/RES value $A_{RR}$;
a set of attributes for mapping onto lifelines $A_L$;
a set of attributes for mapping onto message parameters
$A_M$;
a case ID which defines trace for which it is necessary to
build model $caseId$;
a set of regular expressions for merging diagram
components $L_{RE}$ ;
Output: $U_{HM} = (U_{HD}, CD)$ — hybrid UML model;
begin
    /* Split event log into several
    parts                                        */
    $Log_w, Log_o \leftarrow$ splitEventLog($Log, A_L, A_{RR}$);
    /* Build activity diagrams using
    $\alpha$-algorithm [3]                       */
    $AD \leftarrow$ buildADsAlpha($Log_o$);
    $U_{SD} \leftarrow$
    buildSD($Log_w, AD, L_{RE}, A_L, A_M, A_{RR}, caseId$);
    $CD \leftarrow$ buildCDs($Log_w, A_L, ARR$);
    return $U_{HM}$;
```

Algorithm 1. Building a hybrid UML model U_{HM}

3.2. Interaction abstraction level

This level shows interaction of one selected service or process with its nearest neighbors. For a given service, its nearest neighbors are caller and callee services. A UML sequence diagram does not fully suit for representing this level as well as an activity diagram. In the former case, a UML sequence diagram contains a time perspective on which no relation can be mapped. Thus, this leads us to have a "blind" diagram. In the latter case, it does not support multiple participants which is important for this abstraction level.

We propose to use UML communication diagrams for depicting processes occurring in SOA system at interaction abstraction level. An example of such a diagram for

`Card::Operations` and `Card::OperationData` from an event log example is presented in Figures 5 and 6.

```
Input  : an event log $Log$;
a set of attributes for mapping onto lifelines $A_L$;
an attribute name with REQ/RES value $A_{RR}$;
Output: $Log_w$ — a part of an event log which contains
        interaction between different services;
$Log_o$ — a set of event logs (parts of initial event log).
Each of them contains events related to an individual
service;
Data:  $f : K \rightarrow \mathcal{P}(V)$, where $K$ is a set of keys and
       $\mathcal{P}(V)$ is a set of value sets;

begin
    /* Get lifeline names from an event
    log                                          */
    $K \leftarrow$ getLifelineNames($Log, A_L$);
    for $\sigma \in Log$ do
        $\sigma' \leftarrow \emptyset$;
        /* stack - stack with nested
        events                                   */
        $stack \leftarrow \emptyset$;
        for $i \leftarrow 1$ to $|\sigma|$ do
            $e \leftarrow \sigma[i]$;
            $f$(getLifelineName($e, A_L$)) $\leftarrow$
            $f$(getLifelineName($e, A_L$)) $\bigcup\{e\}$;
            if isRequest($e, A_{RR}$) $= true$ then
                $e_{prev} \leftarrow stack.peek()$;
                if
                $i = 0 \bigvee$ getLifelineName($e, A_L$)! =
                getLifelineName($e_{prev}, A_L$) then
                    $\sigma' \leftarrow \sigma' \bigcup\{e\}$;
                $stack.push(e)$;
            if isRequest($e, A_{RR}$) $= false$ then
                $stack.pop()$;
        $Log_w \leftarrow Log_w \bigcup\{\sigma'\}$;
    for $k \in K$ do
        $Log_o \leftarrow Log_o \bigcup\{f(k)\}$;
    return $Log_w, Log_o$;
```

Algorigm 2. Splitting of an event log into several parts `splitEventLog`

```
Input  : an event log $Log$; a set of UML activity
         diagrams $AD$ which $U_{SD}$ will be refer to;
a set of regular expressions for merging diagram
components $L_{RE}$ a set of attributes for mapping onto
lifelines $A_L$; a set of attributes for mapping onto
message parameters $A_M$; an attribute name with
REQ/RES value $A_{RR}$;
a case ID which defines trace for which it is necessary to
build model $caseId$;
Output: $U_{SD} = (L, T, A, \Lambda, M, Ref, F)$ — UML
         sequence diagram referring to UML activity
         diagrams;

begin
    /* Get lifelines from event log     */
    $L \leftarrow$ mapLifelines($Log, A_L$) ;
    if $caseId = \emptyset$ then
        $isAlt \leftarrow true$;
        $caseId \leftarrow$
        getCaseIdOfLongestTrace($Log$);
    else
        $isAlt \leftarrow false$;
    /* Get trace with case ID which is
    equal to $caseId$                             */
    $\sigma \leftarrow Log[caseId]$;
    for $i \leftarrow 1$ to $|\sigma|$ do
        $e \leftarrow \sigma[i]$;
        while isRequest($e, A_{RR}$) $= true$ do
            if $isAlt = true$ then
                /* Look for differences
                between corresponding events
                in other traces, add found
                events to diagram using
                combined fragments           */
                findFrames($Log, caseId, e, U_{SD}, A_M, A_{RR}$);
            else
                /* Get a message parameter
                and add its message to
                diagram                      */
                mapMessage($e, A_M, M, A, Ref$);
            $i \leftarrow i + 1$;
        while isRequest($e, A_{RR}$) $= false$ do
            if $isAlt = true$ then
                findFrames($Log, caseId, e, U_{SD}$);
            else
                mapResponseMessage($e, A_M, M, A, Ref$);
            $i \leftarrow i + 1$;
    if $L_{RE}! = \emptyset$ then
        /* Merge components of the diagram
        using regular expressions          */
        changeDiagramUsingREs($U_{SD}, L_{RE}$)
    return $U_{SD}$;
```

Algorithm 3. Building a UML sequence diagram `buildSD`

```
Input  : an event log $Log$;
a current event $e$;
a UML sequence diagram
$U_{SD} = (L, T, A, \Lambda, M, Ref, F)$;
a set of attributes for mapping onto message parameters
$A_M$;
an attribute name with REQ/RES value $A_{RR}$;
a case ID which defines trace for which it is necessary to
build model $caseId$;
Data: $Tree$ is a tree with interaction operands

begin
    $equalCases \leftarrow \emptyset$;
    /* Look for corresponding not equal
    events in other traces, group case
    IDs with equal events into $equalCases$
    */
    $notEqEvents \leftarrow$
    findNotEqEvents($e, Log, caseId, equalCases$);
    if $notEqEvents! = \emptyset \bigvee$
    isLastTrace($e, Log$) $= true$ then
        /* Look for operand where it is
        necessary to add events                    */
        $toAdd \leftarrow$ findOperand($equalCases, Tree$);
        addMessagesToFragment($e, equalCases,$
        $toAdd, Tree$);
    else
        if $Tree = \emptyset$ then
            $Tree \leftarrow$ newNode($equalCases$);
        if isRequest($e, A_{RR}$) $= true$ then
            mapMessage($e, A_M, M, A, Ref$);
        else
            mapResponseMessage($e, A_M, M, A, Ref$);
```

Algorithm 4. Looking for differences between corresponding events in other traces `findFrames`

```
Input  : an event log $Log_w$;
a set of attributes for mapping onto lifelines $A_L$;
an attribute name with REQ/RES value $A_{RR}$;
Output: $CD$ — a set of UML communication diagrams
        for each service;

begin
    /* Iterate through lifeline names
    (participants) from an event log    */
    for $l \in$ getLifelines($Log_w, A_L$) do
        $L_{CD} \leftarrow \{l\}$;
        $M_{CD} \leftarrow \emptyset$;
        for $\sigma \in Log$ do
            for $i \leftarrow 1$ to $|\sigma|$ do
                $e \leftarrow \sigma[i]$;
                if $i! = 0 \bigwedge$ getLifeline($e, A_L$) $= l$
                then
                    $l' \leftarrow$ getLifeline($e_{prev}, A_L$);
                    if $l' \notin L_{CD}$ then
                        $L_{CD} \leftarrow L_{CD} \bigcup \{l'\}$;
                        $M_{CD} \leftarrow M_{CD} \bigcup \{(l', l, \emptyset)\}$;
                if $i! = 0 \bigwedge$
                getLifeline($e_{prev}, A_L$) $= l$ then
                    $l' \leftarrow$ getLifeline($e, A_L$);
                    if $l' \notin L_{CD}$ then
                        $L_{CD} \leftarrow L_{CD} \bigcup \{l'\}$;
                        $M_{CD} \leftarrow M_{CD} \bigcup \{(l, l', \emptyset)\}$;
                if isRequest($e, A_{RR}$) $= true$ then
                    $e_{prev} \leftarrow e$
        $U_{CD} \leftarrow (L_{CD}, M_{CD}, \Lambda)$;
        $CD \leftarrow CD \bigcup \{U_{CD}\}$;
    return $CD$;
```

Algorithm 5. Building UML communication diagrams for each service `buildCDs`

3.3. Building process

Figure 7 represents a workflow diagram of a hybrid mining process. The scheme contains the following tasks (see Algorithm 1):

- An event log is split into several parts. The workflow part of the log refers to services communication. Such communication is represented on a UML sequence diagram at workflow level. The operation parts consist of events referred to activity only inside a specific service.
- A UML sequence diagram is built from a workflow part of an event log using the method proposed in [9] (see Section III-D) extended by a number of necessary *ref* fragments used for connecting with corresponding activity diagrams.

- UML activity diagrams are built from the operation parts of the log independently using one of the process mining algorithms which produces a Petri net. For instance, α-algorithm [4] or inductive miner [27] can be considered here. Then, Petri nets are converted into activity diagrams by a simple translation routine. This conversion is rather trivial since UML activity diagrams are initially based on Petri nets [7], [23].

3.4. Mining UML sequence diagrams

To mine a UML sequence diagram we use a method proposed in [9]. There, we propose an approach to mining UML sequence diagrams with different levels of abstraction. It consists of three steps. The first step of the approach is mapping event log attributes onto UML sequence diagram components. There are two functions for mapping attributes onto lifelines and message parameters. The smaller the SOA element we choose for mapping onto lifelines, the lower the abstraction level we receive.

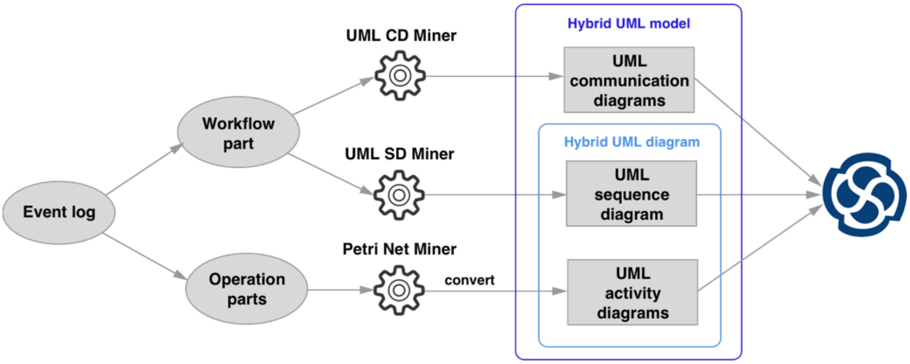


Fig. 7. The workflow diagram of a hybrid mining process.

The second step is set to build a smaller model by applying regular expressions for merging similar messages and lifelines on a diagram. For example, we have two messages with the following parameters: `GetPlaseAndDate, op=BP Billing Transfer` and `GetPlaseAndDate, op=Retail`. They differ in op value, thus, these messages can be combined into one message with the following parameter: `GetPlaseAndDate, op=.*`. After the merging, a derived model becomes more generalized and its size decreases in width and height.

To demonstrate the hierarchy of calls, which is important for SOA, a hierarchical diagram can be applied. Thus, the third step of our approach contains a way to present a complex model by using hierarchical UML diagrams. UML standard [7] allows us to divide the model into some parts and connect them by means of *interaction use* (*ref* fragment) and *gates*.

4. *Tool Overview*

This section presents a brief overview of the software tool implementing the proposed algorithm.

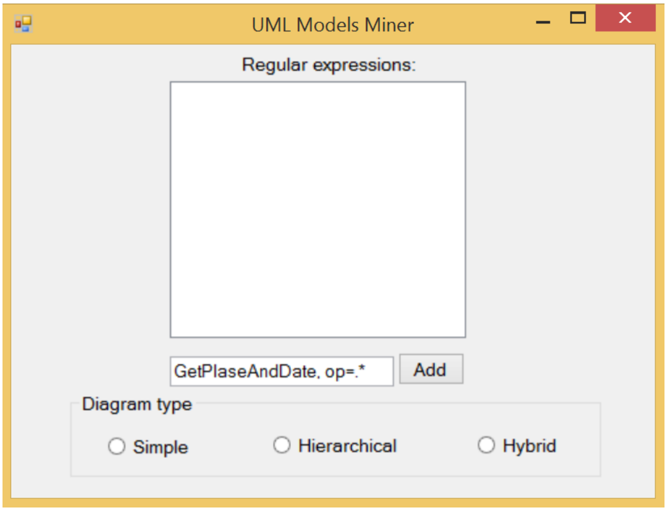
4.1. Event log

The tool requires an input event log to be presented in definite format. We use simple CSV text files to represent event logs. An event log should contain a number of fields that are mapped onto mandatory attributes, namely *CaseID*, *Timestamp* and *Activity*.

4.2. Tool implementation

The tool is implemented as a Windows application written in C# programming language. The tool allows users to configure main parameters such as regular expressions, hierarchy and type of output diagram (regular UML, hierarchical or hybrid). Regular expressions are applied for merging diagram components. It is implemented as shown in Figure 8. The GUI allows the user to set the type of diagram. The *perspective* of the diagram (a mapping attributes onto diagram lifelines and messages) is set as it described in [9].

The output of the tool is an XMI-file containing a model and a description of diagrams. It can be visualized by Sparx Enterprise Architect [15].



GetPlaseAndDate, op=.*

Fig. 8. GUI to set a type of the diagram and regular expressions for merging its components.

5. Conclusion

This paper introduced a new concept of hybrid UML models and proposed a method of mining them from event logs of SOA information systems using a service mining approach. Our method can also be applied to other types of UML diagrams. The paper discussed approaches to mining diagrams at different abstraction levels.

Our method builds models by using only event logs. This is an advantage over some reverse engineering techniques because the source code is not always available. The proposed method includes mining hybrid UML diagrams that represent workflow abstraction level on UML sequence diagrams and operation level on UML activity diagrams. Moreover, we proposed to build UML communication diagrams to show interaction abstraction level with regards to the service mining approach.

Generally, control structures in system‘s behavior lead to a presence of a big number of nested combined fragments within a UML sequence diagram. It makes the diagram less readable and less understandable. Although UML activity diagrams have no time perspective in contradistinction to sequence diagrams, the former show alternatives, loops and parallelism more clearly. Since there are also a lot of event logs which are not produced by SOA systems, we are going to expand our approach to mining hybrid UML diagrams from event logs of more broad types of software architecture in the future.

Acknowledgement

This work is supported by the Basic Research Program at the National Research University Higher School of Economics.

References

[1]. W. M. P. van der Aalst. Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer Publishing Company, Incorporated, 1st edition, 2011.

[2]. V. Rubin, C. W. Günther, W. M. P. van der Aalst, E. Kindler, B. F. van Dongen, and W. Schäfer. Process Mining Framework for Software Processes, pages 169– 181. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[3]. A. K. A. de Medeiros, B. F. van Dongen, W. M. P. van der Aalst, and A. J. M. M. Weijters. Process mining: Extending the α-algorithm to mine short loops. In Eindhoven University of Technology, Eindhoven, 2004.

[4]. W. M. P. van der Aalst, A. J. M. M. Weijter, and L. Maruster. Workflow Mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering, 16:2004, 2003.

[5]. F. Friedrich, J. Mendling, and F. Puhlmann. Process Model Generation from Natural Language Text, pages 482–496. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[6]. C. W. Günther and W. M. P. van der Aalst. Fuzzy Mining – Adaptive Process Simplification Based on Multiperspective Metrics, pages 328–343. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[7]. OMG. OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.5, August 2015.

[8]. T. Erl. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[9]. K. V. Davydova and S. A. Shershakov. Mining Hierarchical UML Sequence Diagrams from Event Logs of SOA systems while Balancing between Abstracted and Detailed Models. 28(3):85–102, 2016.

[10]. S. A. Shershakov and V. A. Rubin. System runs analysis with process mining. In Modeling and Analysis of Information Systems, pages 818–833, 2015.

[11]. A. Rountev and B. H. Connell. Object Naming Analysis for Reverse-engineered Sequence Diagrams. In Proceedings of the 27th International Conference on Software Engineering, ICSE ’05, pages 254–263, New York, NY, USA, 2005. ACM.

[12]. A. Rountev, O. Volgin, and M. Reddoch. Static Control-flow Analysis for Reverse Engineering of UML Sequence Diagrams. SIGSOFT Softw. Eng. Notes, 31(1):96–102, September 2005.

[13]. P. Tonella and A. Potrich. Reverse engineering of the interaction diagrams from C++ code. In International Conference on Software Maintenance, pages 159–168. IEEE Computer Society, 2003.

[14]. E. Korshunova, M. Petkovic, M. G. J. van den Brand, and M. R. Mousavi. CPP2XMI: Reverse Engineering of UML Class, Sequence, and Activity Diagrams from C++ Source Code. In WCRE, pages 297–298. IEEE Computer Society, 2006.

[15]. Sparx Systems’ Enterprise Architect. http://www.sparxsystems.com.au/products/ea/.

[16]. IBM Rational Software Architect. https://www.ibm.com/ developerworks/downloads/r/architect/.

[17]. Visual Paradigm. https://www.visual-paradigm.com/ features/.

[18]. Altova UModel. http://www.altova.com/umodel.html.

[19]. NetBeans UML. http://plugins.netbeans.org/plugin/1801/netbeans-uml.

[20]. L. C. Briand, Y. Labiche, and J. Leduc. Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software. IEEE Trans. Softw. Eng., 32(9):642–663, September 2006.

[21]. R. Delamare, B. Baudry, and Y. Le Traon. Reverse-engineering of UML 2.0 Sequence Diagrams from Execution Traces. In Proceedings of the workshop on Object-Oriented Reengineering at ECOOP 06, Nantes, France, July 2006.

[22]. T. Ziadi, M. A. A. da Silva, L. M. Hillah, and M. Ziane. A Fully Dynamic Approach to the Reverse Engineering of UML Sequence Diagrams. In Isabelle Perseil, Karin Breitman, and Roy Sterritt, editors, ICECCS, pages 107– 116. IEEE Computer Society, 2011.

[23]. B. Agarwal. Transformation of UML Activity Diagrams into Petri Nets for Verification Purposes. 2(3):798–805, 2013.

[24]. A. Bergmayr, H. Bruneliere, J. Cabot, J. García, T. Mayerhofer, and M. Wimmer. fREX: FUML-based Reverse Engineering of Executable Behavior for Software Dynamic Analysis. In Proceedings of the 8th International Workshop on Modeling in Software Engineering, MiSE ’16, pages 20–26, New York, NY, USA, 2016. ACM.

[25]. S. Dustdar, R. Gombotz, and K. Baina. Web Services Interaction Mining. Tech. Rep. TUV-1841-2004-16. 2004.

[26]. W. M. P. van der Aalst. Service Mining: Using Process Mining to Discover, Check, and Improve Service Behavior. IEEE Transactions on Services Computing, 6(4):525–535, 2013.

[27]. S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst. Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour, pages 66–78. Springer International Publishing, Cham, 2014.

Метод автоматического построения гибридных UML-моделей на основе журналов событий систем с сервис-ориентированной архитектурой

К.В. Давыдова <kvdavydova@edu.hse.ru>
С.А. Шершаков <sshershakov@hse.ru>
Национальный исследовательский университет Высшая школа экономики, лаборатория ПОИС факультета компьютерных наук,
101000, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. В данной статье мы предлагаем метод автоматического построения так называемых «гибридных» UML-моделей, что относится к области извлечения и анализа процессов ПО. Модели строятся на основе трасс исполнения, представленных в виде журналов событий, систем с сервис-ориентированной архитектурой (СОА). В то время как известные техники обратной разработки обычно используют исходный программный код, который часто недоступен, наш подход работает с журналами событий, записываемыми большинством информационных систем, и некоторыми эвристическими параметрами. Так как отдельный класс UML-диаграмм представляет только одну перспективу модели системы, мы предлагаем синтезировать комбинацию нескольких классов UML-диаграмм (последовательности и деятельности), которые рассматриваются совместно с диаграммами коммуникаций. Это позволяет повысить выразительную силу отдельной «гибридной» диаграммы. Каждый класс диаграмм представляет один из уровней абстракции (workflow, operation и interaction), которые обычно используются при рассмотрении взаимодействия web-сервисов. Предлагаемый алгоритм состоит из четырех этапов: разделение журнала событий на несколько частей, построение UML диаграмм последовательности, деятельности и коммуникаций. Мы также предлагаем инкапсулировать некоторые незначительные или низкоуровневые имплементационные детали (например, внутренние операции сервисов) в диаграммы деятельности и соединять их с более высокоуровневыми диаграммами последовательности с использованием «interaction use» фрагментов. Чтобы решить проблему больших размеров синтезируемых UML диаграмм последовательности, мы предлагаем обобщающую технику, основанную на регулярных выражениях. Предложенный подход оценен с использованием разработанного программного инструмента в виде Windows-приложения, написанного на языке C#. Этот инструмент строит UML модели и сохраняет их в виде XML-файлов. Такие файлы совместимы с хорошо известным интрументом проектирования программной архитектуры Sparx Enterprise Architect, в котором синтезированные модели могут быть визуализированы и отредактированы.

Ключевые слова: журнал событий; извлечение и анализ процессов (process mining); гибридные UML модели; диаграмма последовательности UML; диаграмма деятельности UML; обратная разработка.

DOI: 10.15514/ISPRAS-2017-29(4)-10

Для цитирования: Давыдова К.В., Шершаков С.А. Метод автоматического построения гибридных UML моделей на основе журналов событий СОА-систем. Труды ИСП РАН, том 29, вып. 4, 2017 г., стр. 155-174 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-10

Список литературы

[1]. W. M. P. van der Aalst. Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer Publishing Company, Incorporated, 1st edition, 2011.

[2]. V. Rubin, C. W. Günther, W. M. P. van der Aalst, E. Kindler, B. F. van Dongen, and W. Schäfer. Process Mining Framework for Software Processes, pages 169– 181. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[3]. A. K. A. de Medeiros, B. F. van Dongen, W. M. P. van der Aalst, and A. J. M. M. Weijters. Process mining: Extending the α-algorithm to mine short loops. In Eindhoven University of Technology, Eindhoven, 2004.

[4]. W. M. P. van der Aalst, A. J. M. M. Weijter, and L. Maruster. Workflow Mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering, 16:2004, 2003.

[5]. F. Friedrich, J. Mendling, and F. Puhlmann. Process Model Generation from Natural Language Text, pages 482–496. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[6]. C. W. Günther and W. M. P. van der Aalst. Fuzzy Mining – Adaptive Process Simplification Based on Multiperspective Metrics, pages 328–343. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[7]. OMG. OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.5, August 2015.

[8]. T. Erl. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[9]. K. V. Davydova and S. A. Shershakov. Mining Hierarchical UML Sequence Diagrams from Event Logs of SOA systems while Balancing between Abstracted and Detailed Models. 28(3):85–102, 2016.

[10]. S. A. Shershakov and V. A. Rubin. System runs analysis with process mining. In Modeling and Analysis of Information Systems, pages 818–833, 2015.

[11]. A. Rountev and B. H. Connell. Object Naming Analysis for Reverse-engineered Sequence Diagrams. In Proceedings of the 27th International Conference on Software Engineering, ICSE ’05, pages 254–263, New York, NY, USA, 2005. ACM.

[12]. A. Rountev, O. Volgin, and M. Reddoch. Static Control-flow Analysis for Reverse Engineering of UML Sequence Diagrams. SIGSOFT Softw. Eng. Notes, 31(1):96–102, September 2005.

[13]. P. Tonella and A. Potrich. Reverse engineering of the interaction diagrams from C++ code. In International Conference on Software Maintenance, pages 159–168. IEEE Computer Society, 2003.

[14]. E. Korshunova, M. Petkovic, M. G. J. van den Brand, and M. R. Mousavi. CPP2XMI: Reverse Engineering of UML Class, Sequence, and Activity Diagrams from C++ Source Code. In WCRE, pages 297–298. IEEE Computer Society, 2006.

[15]. Sparx Systems' Enterprise Architect. http://www.sparxsystems.com.au/products/ea/.

[16]. IBM Rational Software Architect. https://www.ibm.com/ developerworks/downloads/r/architect/.

[17]. Visual Paradigm. https://www.visual-paradigm.com/ features/.

[18]. Altova UModel. http://www.altova.com/umodel.html.

[19]. NetBeans UML. http://plugins.netbeans.org/plugin/1801/netbeans-uml.

[20]. L. C. Briand, Y. Labiche, and J. Leduc. Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software. IEEE Trans. Softw. Eng., 32(9):642–663, September 2006.

[21]. R. Delamare, B. Baudry, and Y. Le Traon. Reverse-engineering of UML 2.0 Sequence Diagrams from Execution Traces. In Proceedings of the workshop on Object-Oriented Reengineering at ECOOP 06, Nantes, France, July 2006.

[22]. T. Ziadi, M. A. A. da Silva, L. M. Hillah, and M. Ziane. A Fully Dynamic Approach to the Reverse Engineering of UML Sequence Diagrams. In Isabelle Perseil, Karin Breitman, and Roy Sterritt, editors, ICECCS, pages 107– 116. IEEE Computer Society, 2011.

[23]. B. Agarwal. Transformation of UML Activity Diagrams into Petri Nets for Verification Purposes. 2(3):798–805, 2013.

[24]. A. Bergmayr, H. Bruneliere, J. Cabot, J. García, T. Mayerhofer, and M. Wimmer. fREX: FUML-based Reverse Engineering of Executable Behavior for Software Dynamic Analysis. In Proceedings of the 8th International Workshop on Modeling in Software Engineering, MiSE '16, pages 20–26, New York, NY, USA, 2016. ACM.

[25]. S. Dustdar, R. Gombotz, and K. Baina. Web Services Interaction Mining. Tech. Rep. TUV-1841-2004-16. 2004.

[26]. W. M. P. van der Aalst. Service Mining: Using Process Mining to Discover, Check, and Improve Service Behavior. IEEE Transactions on Services Computing, 6(4):525–535, 2013.

[27]. S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst. Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour, pages 66–78. Springer International Publishing, Cham, 2014.

Tool for Behavioral Analysis of Well-Structured Transition Systems

L.W. Dworzanski <leo@mathtech.ru>
V.E. Mikhaylov <vlamikhaylov@gmail.com>
Department of Software Engineering,
National Research University Higher School of Economics,
Myasnitskaya st., 20, Moscow, 101000, Russia

Abstract. Well-structured transition systems (WSTS) became a well-known tool in the study of concurrency systems for proving decidability of properties based on coverability and boundedness. Each year brings new formalisms proven to be WSTS systems. Despite the large body of theoretical work on the WSTS theory, there has been a notable gap of empirical research of well-structured transition systems. In this paper, the tool for behavioural analysis of such systems is presented. We suggest the extension of SETL language to describe WSTS systems (WSTSL). It makes the description of new formalisms very close to the formal definition. Therefore, it is easy to introduce and modify new formalisms as well as conduct analysis of the behavioural properties without much programming efforts. It is highly convenient when a new formalism is still under active development. Two most studied algorithms for analysis of well-structured transition systems behavior (backward reachability and the Finite Reachability Tree analyses) have been implemented; and, their performance was measured through the runs on such models as Petri Nets and Lossy Channel Systems. The developed tool can be useful for incorporating and testing analysis methods to formalisms that occur to be well-structuredness transition systems.

Keywords: formal verification; infinite systems; well structured transition systems; Petri nets

DOI: 10.15514/ISPRAS-2017-29(4)-11

For citation: Dworzanski L.V., Mikhaylov V.E. Tool for Behavioral Analysis of Well-Structured Transition Systems. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 175-190. DOI: 10.15514/ISPRAS-2017-29(4)-11

1. Introduction

Formal verification provides researchers and developers with approaches that are widely-used for proving that a program satisfies a formal specification of its behavior. These methods are highly demanded in the software and hardware engineering, as they provide appropriate level of systems reliability; which, in most cases, cannot be ensured by simulation.

One of the most common technique of formal verification is model checking or property checking. It involves algorithmic methods that are applied to check satisfiability of a logic formula used for the representation of the specification and the model of a system. The main advantage of model checking is considered to be the fact that it enables almost completely automatic process of verification. Model checking proved to be effective in practice for analysis of finite-state systems [1]; however, in case of systems with infinite state space the situation is more complicated because exhaustive search, which is usually used by verification tools, cannot be applied directly.

In order to deal with infinite-state systems Finkel proposed the idea of well-structured transition systems (WSTS) in 1987 [2]. “These are transition systems where the existence of a well-quasi-ordering over the infinite set of states ensures the termination of several algorithmic methods. [3]” The suggested model has provided researchers with an abstract generalization of several models (e.g. Petri nets, lossy channel systems and timed automata). Therefore, the results obtained from the analysis of such a generalized model can be also applied to these specific models.

The WSTS analysis can be used to solve, for instance, covering, termination, inevitability and boundedness problems. However, the application of the WSTS analysis is hampered by the necessity of implementing algorithms and data structures to support the analysis for each new formalism. In this work, the tool that can be used for analysis of WSTS is presented. We introduce the WSTSL language - modification of SETL language [13,14] – set-theoretical programming language. The language provides the user with opportunity to define the structure of analyzed system as close to the original formal definition as possible. After definition of the formalism, it is immediately possible to run backward reachability method [4] or the Finite Reachability Tree [5] on it. It is convenient for computer science researcher to postpone the implementation phase after what-if experiments.

The rest of the paper is organized as follows. The second section describes WSTS’s basic terms and underlying concepts. The third section provides the description of two used algorithms (the backward reachability method and the Finite Reachability Tree). The forth section presents the architecture of the developed analysis tool. The fifth section shows how the developed tool is used for the analysis of Petri nets and provides performance analysis results. The sixth section summarizes and provides possible applications of the study for the future research.

2. Well-Structured Transition Systems

The definition of well-structured transition systems (WSTS) was proposed by Finkel in [2]. It is based on the two main concepts: transition systems (TS) and well-quasi-orderings between the states of these systems.

Transition system (TS) is one of the most widely-used models for formal description of the behavior of different systems. A *transition system* is defined by a structure $TS = (S, \rightarrow, \dots)$ where $S = \{s, t, \dots\}$ is a set of *states*, and $\rightarrow \subseteq S \times S$ is any set of *transactions* [3]. TS can be also supplemented by other structures such as initial states, labels for transitions, durations or causal independence relations [3]; however, for the consideration of the concept of WSTS using of set of states along with set of transactions is sufficient.

A binary relation $\leq$ on a set X is called *preorder* or *quasi-ordering (qo)* if it is reflexive and transitive. So for any $a, b, c \subseteq X$ we have:

1) $a \leq a$ (reflexivity);
2) $if\ a \leq b\ and\ b \leq c\ then\ a \leq c$ (transitivity).

Definition 1. A *well-quasi-ordering (wqo)* is a qo in which for every infinite sequence of elements $x_0, x_1, x_2, x_3, \dots \subseteq X$ there exist such indices $i < j$ that $x_i \leq x_j$ [3, 6]. According to [7], there are a range of equal definitions of wqo; however, the definition given above is generally used in papers on WSTS.

Definition 2. A *well-structured transition system* (WSTS) is a transition system T$S = (S, \rightarrow, \leq)$ equipped with a qo $\leq \subseteq S \times S$ between states such that the two following conditions hold:

1) well-quasi-ordering: $\leq$ is a wqo, and
2) compatibility: $\leq$ is (upward) compatible with $\rightarrow$, i.e. for all $s_1 \leq t_1$ and transition $s_1 \rightarrow s_2$, there exists such a sequence of transitions $t_1 \rightarrow^* t_2$ that $s_2 \leq\ t_2$ [3].

$Succ(s)$ denotes the set $\{s' \in S \mid s \rightarrow s'\}$ of immediate successors of s. Likewise, $Pred(s)$ denotes the set $\{s' \in S \mid s' \rightarrow s\}$ of immediate predecessors.

An upward-closed set is any set $I \subseteq X$ such that $y \geq x$ and $x \in I$ entail $y \in I$. A basis of an upward-closed I is a set I^b such that $I = \cup_{x \in I^b} \uparrow x$, where $\uparrow x =^{def} \{y \mid y \geq x\}$.

3. Algorithms

3.1 Backward Reachability Method

Backward reachability method proposed by Abulla et al. in [4] is intended to solve the covering problem which is to decide, given two states s and t, whether starting from s it is possible to reach a state $t' \geq t$. This is essentially one of set-saturation methods termination of which relies on the lemma that says that any increasing sequence of upward-closed sets ($I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots$) eventually stabilizes (i.e. there is such a $k \in N$ that $I_k = I_{k+1} = I_{k+2} = \cdots$) [3].

Assume there is some WSTS $TS = (S, \rightarrow, \leq)$ and some upward-closed set of states $I \subseteq S$. Backward reachability method on the each j-th step generates the set of states from which I can be reached by a sequence at most j transitions [4].

More strict generalization was suggested by Finkel and Schnoebelen in [3], where it involves computing $Pred^*(I)$ as the limit of the sequence $I_0 \subseteq I_1 \subseteq \cdots$ where $I_0 =^{def} I$ and $I_{n+1} =^{def} I_n \cup Pred(I_n)$.

Definition 3. A WSTS has effective pred-basis if there exists an algorithm accepting any state $s \in S$ and returning $pb(s)$, a finite basis of $\uparrow Pred(\uparrow s)$.

The covering problem is decidable for WSTS if it has effective pred-basis and decidable $\leq$. The proof of this statement is given in [3]. Essentially, it is said that if there is a sequence K_0, K_1 ... with $K_0 =^{def} I^b$ (finite basis of I), $K_{n+1} =^{def} K_n \cup pb(K_n)$ and m is the first index such that $\uparrow K_m = \uparrow K_{m+1}$, then $\uparrow \cup K_i = Pred^*(I)$. By decidability of $\leq$, it is possible to check whether $s \in \uparrow Pred^*(\uparrow t)$.

3.2 Finite Reachability Tree

The Finite Reachability Tree belongs to tree-saturation methods which represent methods that consider all possible computations inside a finite tree-like structure [3]. It is also called the forward analysis method, in contrast to the backward analysis. Essentially, it is based on the ideas proposed by Karp and Miller in [5].

Assume there is some WSTS $TS = (S, \rightarrow, \leq)$. For any state $s \in S$, the Finite Reachability Tree is such a finite directed graph (tree) that:

1) nodes of the tree are labeled by states of S;
2) nodes are either dead or live;
3) the root node is a live node n_0, labeled by s (written $n_0 : s$);
4) dead nodes have no child nodes;
5) a live node $n : t$ has one child $n' : t'$ for each successor $t' \in Succ(t)$;
6) if along the path from the root $n_0 : s$ to some node $n' : t'$ there exists a node $n : t$ $(n \neq n')$ such that $t \leq t'$, we say that n subsumes n', and then n' is a dead node [3, 6].

The Finite Reachability Tree is effectively computable if S has (1) a decidable $\leq$, and (2) $Succ$ mapping is computable [3]. All paths in the finite reachability tree are finite as any infinite path would include a covering node [6].

This algorithm can be applied to termination, inevitability, and boundedness problems (see [3] for details).

4. Proposed Architecture

The general structure of the architecture of the developed tool is illustrated in Fig. 1. It consists of two main parts: Well-Structured Transition Systems Language

(WSTSL) and WSTS Analyzer. Also there are four input parameters that are set by the user through WSTSL.

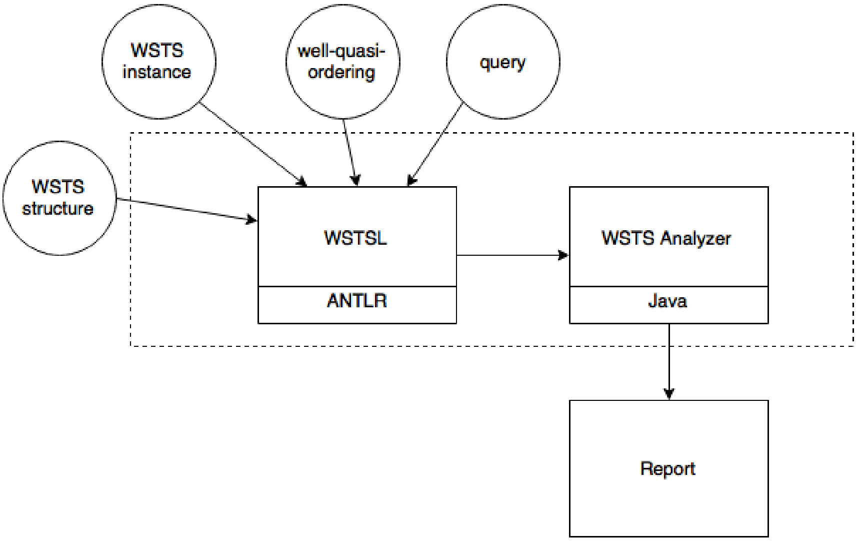


Fig. 1. Architecture of the developed tool

WSTSL is a programming language used in the developed system as the front-end which provides user with a means of describing input parameters. Therefore, the following data types are included: integers, tuples, maps and sets. To run the appropriate algorithm the user has to use either *backwardanalysis()* or *forwardanalysis()* command. As it is depicted in Fig.1 the parser for WSTSL is built with Another Tool for Language Recognition (ANTLR), which generates it from a formal language description called a grammar [8]. The parser's sources are generated in Java, since ANTLR itself is written in Java and provides more parsing capabilities for some cases in comparison with other supported target languages (C#, JavaScript, Python2, Python3, Swift, Go).

WSTS Analyzer represents that part of the system which is responsible for the processing of the input transition system, which it gets from the WSTSL parser, and the application of the algorithm selected by the user. WSTS Analyzer is implemented in Java, as it allows running it in all platforms that support Java, and, most importantly, naturally interacts with parser's Java classes generated by ANTLR.

As it was noted above, the input that is provided by the user includes four main parts. Firstly, a general structure (WSTS structure) of the analyzed transition system should be described (e.g. Petri nets or lossy channel systems in general). Secondly, a well-quasi-ordering should be specified. Then, a structure of a specific transition system (WSTS instance) that corresponds to the general structure is provided.

Finally, the desired analysis algorithm with appropriate parameters (query) is specified. Essentially, all these parts are described in a single input program written in WSTSL. Afterwards, the WSTS Analyzer runs the selected algorithm on the specified system and generates report which format depends on the choice of the algorithm.

5. Experiment

5.1 Petri Net

The applicability of the proposed approach could be demonstrated by an example with common well-structured transition system called Petri net. The classical definition of this model is the following.

Definition 4. A Petri net (P/T-net) is a 4-tuple (P, T, F, W) where

- P and T are disjoint finite sets of places and transitions, respectively;
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs;
- $W : F \rightarrow \mathbb{N} \setminus \{0\}$ – an arc multiplicity function, that is, a function which assigns every arc a positive integer called an arc multiplicity or weight.
- A marking of a Petri net (P, T, F, W) is a multiset over P, i.e. a mapping $M : P \rightarrow \mathbb{N}$. By $M(N)$ we denote the set of all markings of the P/T-net N.
- We say that a transition t in the P/T-net $N = (P, T, F, W)$ is active in marking M if for every $p \in \{p \mid (p, t) \in F\}$: $M(p) \geq W(p, t)$. An active transition may fire, resulting in a marking M', such as for all $p \in P$: $M'(p) = M(p) - W(p, t)$ if $p \in \{p \mid (p, t) \in F\}$, $M'(p) = M(p) - W(p, t) + W(t, p)$ if $p \in \{p \mid (t, p) \in F)$ and $M'(p) = M(p)$ otherwise.

For simplicity's sake, we consider here the Petri net which arcs can only have multiplicity 1.

For the experiment the Petri net illustrated in Fig. 2 will be considered.

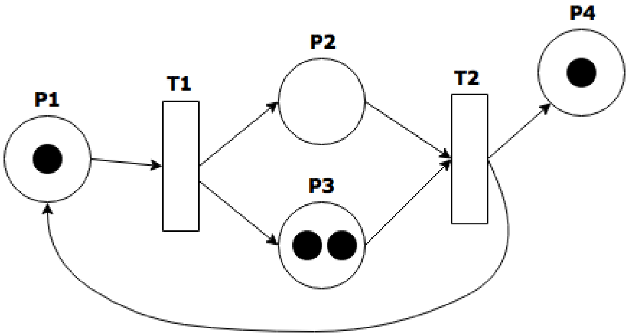


Fig. 2. Instance of the Petri net for consideration in the experiment

First of all, the general structure of the Petri net model described above should be defined by means of WSTL (Fig. 3).

```
type P                 : set of int;
type T                 : set of int;
type PT(P1:P, T1:T)    : set of [from P1,from T1];
type TP(P1:T, P1:P)    : set of [from T1,from P1];
type M(P1:P)           : map <from P1,int>;
type PN(P1:P, T1:T,
        PT1:PT, TP1:TP) : [P1,T1,PT1,TP1];
```

Fig. 3. General structure of Petri net in WSTSL

Secondly, we describe the specific Petri net instance in WSTSL (Fig. 4). PT1 and TP1 represent the arcs from places to transitions and vice versa, respectively. In tuples, defining arcs, the corresponding transition goes first for the convenience in description of *Succ* and *Pred* function as it will be seen below.

```
var P1:P = {"P1","P2","P3","P4"};
var T1:T = {"T1","T2"};
var PT1:PT(P1,T1) = {["T1","P1"],["T2","P2"],
                     ["T2","P3"]};
var TP1:TP(T1,P1) = {["T1","P2"],["T1","P3"],
                     ["T2","P1"],["T2","P4"]};
```

Fig. 4. Description of the specific Petri net instance in WSTSL

Then, a well-quasi-ordering should be described (Fig. 5). As it is shown in [3], the inclusion ordering ($M \subseteq M'$when $M(p) \leq M'(p)$ for every place) is a wqo and it is known as Dickson's lemma [9]. Operator *forall iterator* | *test* generates a boolean value *true* if the condition *test* is met for each step in *iterator* and a boolean value *false* otherwise.

```
func wqo(PN1:PN, s1:M, s2:M)
    return forall p in PN[0] | s1[p] <= s2[p];
end func;
```

Fig. 5. Well-quasi-ordering function described as inclusion ordering in WSTSL

As it has been mentioned above in the Algorithms section, Backward Reachability Method requires effective algorithm for computation of *pred-basis*. The algorithm to compute it for Petri Net was suggested in [4]. How it is described in WSTSL is shown in Fig. 6.

```
func pred(PN1:PN, K:set of M)
    var P1:P = PN1[0];
    var T1:T = PN1[1];
    var PT1:PT(P1,T1) = PN1[2];
    var TP1:TP(T1,P1) = PN1[3];
    var predecessors: set of M(P1) = { };

    for s in K
        for t in T
            if forall tp in TP1[t] | s[tp[1]] - 1 >= 0 then
                s1 = s;
                for pt in PT1[t]
                    s1[pt[1]] = s1[pt[1]] + 1;
                end for;
                for tp in TP1[t]
                    s1[tp[1]] = s1[tp[1]] - 1;
                end for;
                predecessors = predecessors with s1;
            end if;
        end for;
    end for;
    return predecessors;
end func;

func pb(PN1:PN, K:set of M)
    return min(pred(PN1, I), wqo)
end func;
```

Fig. 6. Description of the pred-basis and pred functions in WSTSL

To solve the covering problem the initial state and the state which coverability it is required to check should be specified. Afterwards, *backwardanalysis* function should be invoked with appropriate arguments (Fig. 7).

```
var m0:M(P1) = {<"P1",1>,<"P2",0>,
                <"P3",2>,<"P4",1>};
var mc:M(P1) = {<"P1",1>,<"P2",1>,
                <"P3",1>,<"P4",2>};

backwardanalysis(PN1,wqo,pb,m0,mc);
```

Fig. 7. Description of the initial marking and the marking which coverability it is required to check with Backward Reachability Method invocation

The tool provides the user with the output that contains sequence of sets K_i, where $K_0 = \{m_c\}$, $K_{n+1} = pb(K_n)$, their union $\cup_{i\in\mathbb{N}} K_i$ and its minimal elements (basis). Finally, it is reported whether the analyzed state (marking) m_c is covered or not (Fig. 8).

```
K0:  [{P1=1,P2=1,P3=1,P4=2}]
K1:  [{P1=0,P2=2,P3=2,P4=1},
      {P1=2,P2=0,P3=0,P4=2}]
K2:  [{P1=1,P2=1,P3=1,P4=1}]
K3:  [{P1=0,P2=2,P3=2,P4=0},
      {P1=2,P2=0,P3=0,P4=1}]
K4:  [{P1=1,P2=1,P3=1,P4=0}]
K5:  [{P1=2,P2=0,P3=0,P4=0}]
Union: [{P1=0,P2=2,P3=2,P4=0},
        {P1=0,P2=2,P3=2,P4=1},
        {P1=1,P2=1,P3=1,P4=0},
        {P1=1,P2=1,P3=1,P4=1},
        {P1=1,P2=1,P3=1,P4=2},
        {P1=2,P2=0,P3=0,P4=0},
        {P1=2,P2=0,P3=0,P4=1},
        {P1=2,P2=0,P3=0,P4=2}]
min(Union): [{P1=0,P2=2,P3=2,P4=0},
             {P1=1,P2=1,P3=1,P4=0},
             {P1=2,P2=0,P3=0,P4=0}]

The state {P1=1,P2=1,P3=1,P4=2} is not covered
```

Fig. 8. Report of the tool for the backward analysis invocation

As it has been mentioned above in the Algorithms section, Finite Reachability Tree requires effective algorithm for computation of *Succ*. How it is described in WSTSL is shown in Fig. 9.

To construct Finite Reachability Tree only the initial state should be specified. Afterwards, *forwardanalysis* function should be invoked with appropriate arguments (Fig. 10).

```
func succ(PN1:PN, s:M)
    var P1:P = PN1[0];
    var T1:T = PN1[1];
    var PT1:PT(P1,T1) = PN1[2];
    var TP1:TP(T1,P1) = PN1[3];
    var successors : set of M(P1) = { };

    for t in T
        if forall pt in PT1[t] | s[pt[1]] - 1 >= 0 then
            s1 = s;
            for pt in PT1[t]
                s1[pt[1]] = s1[pt[1]] - 1;
            end for;
            for tp in TP1[t]
                s1[tp[1]] = s1[tp[1]] + 1;
            end for;
            successors = successors with s1;
        end if;
    end for;
    return successors;
end func;
```

Fig. 9. Description of the Succ function in WSTSL

```
var m0:M(P1) = {<"P1",1>,<"P2",0>,
                <"P3",2>,<"P4",1>};

forwardanalysis(PN1,wqo,succ,m0);
```

Fig. 10. Description of the initial marking and the Finite Reachability Tree construction invocation

The tool provides the user with the image which illustrates constructed Finite Reachability Tree (Fig. 11). Nodes are labeled with their states. Dead nodes are red. The node labeled with {P1=1, P2=0, P3=2, P4=2} state is dead since {P1=1, P2=0, P3=2, P4=2} ≥{P1=1, P2=0, P3=2, P4=1} (the latter state is represented by the root which subsumes the dead node labeled by the former state).

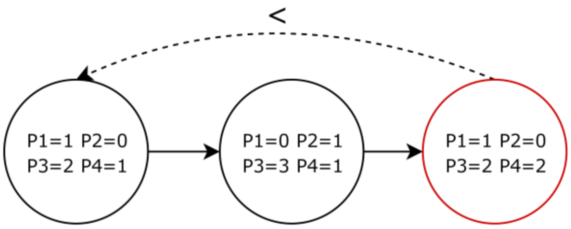


Fig. 11. Constructed finite reachability tree

5.2. Lossy Channel Systems

Another model that we considered was Lossy channel system (LCS) which is a subclass of FIFO-channel systems.

Definition 5. FIFO-channel system is a 6-tuple $(S, s_0, A, C, M, \delta)$ where

- S is a finite set of control states;
- $s_0 \in S$ is the initial control state;
- A is a finite set of actions;
- C is a finite set of channels;
- M is a finite set of messages (M^* is a set of finite strings composed of elements from M);
- δ is a finite set of transitions, each of which is represented by one of the following tuples $(s_1, c!m, s_2)$, $(s_1, c?m, s_2)$, (s_1, a, s_2), where $s_1, s_2 \in S$, $c \in C$, $m \in M$ and $a \in A$ (see below).

Transition $(s_1, c!m, s_2)$ changes the control state from s_1 to s_2, adding the message m to the end of the channel c. Operation $c!m$ is also known as a send action.

Transition $(s_1, c?m, s_2)$ changes the control state from s_1 to s_2, removing the message m from the beginning of the channel c. If the channel c is empty or its first element is not m, then this transition cannot occur. Operation $c?m$ is also known as a receive action.

Transition $(s_1, c?m, s_2)$ changes the control state from s_1 to s_2 and does not change the state of the channels.

Considering LCS it is also assumed that some message in some channel can be lost at any moment. To model this behavior one more operation $\tau(c, m)$ is introduced.

Transition $(s_1, \tau(c, m), s_2)$ removes the message m from the channel c, and does not change the control state.

For LCS = $(S, s_0, A, C, M, \delta)$ the ordering $\leq$ is defined on the set of global states $\{(s, w) | \, s \in S, w: C \to M^*\}$ as follows:

$$(s, w) \leq (s', w') \Leftrightarrow s = s' \wedge w(c) \ll w'(c) \; \forall c \in C.$$

The ordering $\ll$ is a subword ordering: $u \ll v$ iff u can be obtained by erasing letters from v. It is shown in [6] that this ordering is a wqo.

The concrete model that we considered was Alternating Bit Protocol (ABP). It is represented by Sender and Receiver which communicate via two FIFO-channels c_M and c_A. Sender sends messages to Receiver via c_M, while Receiver sends acknowledgements via c_A. Both channels can lose messages. Messages and acknowledgements contain one-bit sequence number 0 or 1. Sender continuously sends the same message with the same sequence number, until it receives an acknowledgement from Receiver with the same sequence number. Then, Sender changes (flips) the sequence number and proceeds with sending the next message. Receiver starts by waiting the message with the sequence number 0 (actually, it can initially send acknowledgments with the sequence number 1). When it receives such a message it starts sending acknowledgements with the same sequence number, until it receives the message with the flipped sequence number and so on. The described model is illustrated in terms of Lossy Channel System in Fig. 12.

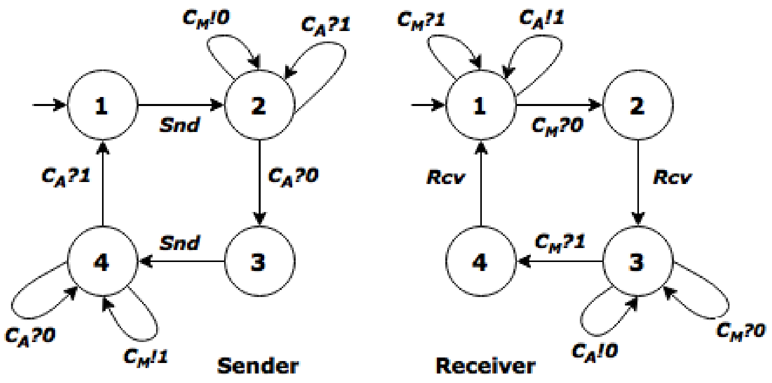


Fig. 12. Alternating Bit Protocol modelled as a Lossy Channel System

5.3 Performance

To measure the performance of the implemented Finite Reachability Tree algorithm we applied it to the four different models, which include a model shown in Fig. 2 (Example 1) and the Petri Net models simulating the dining philosophers problem [10] for a number of philosophers equal to 5, 6 and 7. We executed the experiment on the following machine: Intel Core i7, 2.22 GHz, 16 GB RAM running OS X El Capitan (v. 10.11.6). *System.nanoTime()* method was invoked immediately before of the beginning of construction of a FRT and immediately after the end of construction, then the difference was calculated to measure run time for one run. In Table 1 in the Run time column average results for 20 runs are given in seconds. As well, sizes of the constructed FRTs are given. It can be seen that both run time and size of FRT grow exponentially for the philosophers problem.

Table 1. Performance of the tool during Philosophers problem solving

	Run time (s)	Size of FRT
Example 1	0.03596	3
Phil5	0.08587	241
Phil6	1.87815	25711
Phil7	5221.64756	88062003

6. Summary

This paper addresses a lack of practical results in studies of well-structured transition systems. In order to fill this gap, there was presented one of the possible ways for development of the system capable to analyze WSTS with two common algorithms: backward reachability method and the Finite Reachability Tree. Well-Structured Transition Systems Language is introduced as a means of describing the user's input, which consists of the description of transition system's structure in general and specific instance's relations and values.

The tool can be used by researchers to investigate the efficiency of the implemented algorithms. It is expected that it is appropriate for conducting experiments on small and mediumsized WSTS. The technology eases the efforts required to check the potential of the WSTS analysis algorithms for practical applications and to make what-if experiments on newly developed formalisms.

The application of the tool is illustrated for the Petri nets and Lossy Channel System formalisms. Also, there were given results of the experiment on Petri nets modeling the dining philosophers problem. The performance analysis of the Finite Reachability Tree applied to this problem demonstrated the expected exponential growth of execution time; and, it indicates the need for further investigations of optimizations (e.g. reduction rules) that can be applied to make the algorithm effectively applicable in practice.

7. Acknowledgements

This work is supported by the Basic Research Program at the National Research University Higher School of Economics and Russian Foundation for Basic Research, project No. 16- 01-00546.

References

[1]. J. Burch, E. Clarke, K. McMillan, D. Dill and L. Hwang, "Symbolic model checking: 1020 States and beyond", *Information and Computation*, vol. 98, no. 2, pp. 142-170, 1992.

[2]. A. Finkel, "Well structured transition systems," Univ. Paris-Sud, Orsay, France, Res. Rep. 365, Aug. 1987.

[3]. A. Finkel and P. Schnoebelen, "Well-structured transition systems everywhere!", *Theoretical Computer Science*, vol. 256, no. 1-2, pp. 63-92, 2001.

[4]. P. Abdulla, K. Čerāns, B. Jonsson and Y. Tsay, "Algorithmic Analysis of Programs with Well Quasi-ordered Domains", *Information and Computation*, vol. 160, no. 1-2, pp. 109-127, 2000.

[5]. R. Karp and R. Miller, "Parallel program schemata", *Journal of Computer and System Sciences*, vol. 3, no. 2, pp. 147-195, 1969.

[6]. E. Kouzmin and V. Sokolov, *Well-Structured Labeled Transition Systems,* Moscow: Fizmatlit, 2005.

[7]. J. Kruskal, "The theory of well-quasi-ordering: A frequently discovered concept", *Journal of Combinatorial Theory, Series A*, vol. 13, no. 3, pp. 297-305, 1972.

[8]. T. Parr, *The definitive ANTLR 4 reference*, Raleigh, NC and Dallas, TX: The Pragmatic Bookshelf, 2013.

[9]. L. Dickson, "Finiteness of the Odd Perfect and Primitive Abundant Numbers with n Distinct Prime Factors", *American Journal of Mathematics*, vol. 35, no. 4, pp. 413-422, 1913.

[10]. E. Dijkstra, "Hierarchical ordering of sequential processes", *Acta Informatica*, vol. 1, no. 2, pp. 115-138, 1971.

[11]. S. Akshay, B. Genest, L. Hélouët, Decidable Classes of Unbounded Petri Nets with Time and Urgency. In: F. Kordon, D. Moldt (eds) Application and Theory of Petri Nets and Concurrency. PETRI NETS 2016. *Lecture Notes in Computer Science, vol 9698*. Springer, Cham

[12]. L. W. Dworzanski, Consistent Timed Semantics for Nested Petri Nets with Restricted Urgency, in: *Formal Modeling and Analysis of Timed Systems Vol. 9884*. Switzerland : Springer International Publishing, 2016. doi Ch. 1. pp. 3-18.

[13]. J. T. Schwartz, "Set Theory as a Language for Program Specification and Programming". Courant Institute of Mathematical Sciences, New York University, 1970.

[14]. R. Dewar, "SETL and the Evolution of Programming." In From Linear Operators to Computational Biology, pp. 39-46. Springer London, 2013.

Инструмент для анализа поведения вполне структурированных систем переходов

Л.В. Дворянский <leo@mathtech.ru>
В.Е. Михайлов <vlamikhaylov@gmail.com>
Национальный исследовательский университет
«Высшая школа экономики»,
101000, Россия, Москва, ул. Мясницкая, 20

Аннотация. Вполне структурированные системы переходов являются хорошо известным инструментом для доказательства разрешимости свойств покрываемости и ограниченности. Каждый год появляются новые формализмы, которые оказываются вполне структурированными системами переходов. Несмотря на большой объем теоретической работы, существует большая потребность в эмпирических изучении вполне структурированных систем переходов. В данной работе представлен инструмент для анализа таких систем. Мы предлагаем расширение высокоуровневого языка SETL для описания вполне-структурированных систем переходов. Это позволяет описывать новые формализмы близко к их формальному определению. Таким образом упрощается создание и изменение новых формализмов, а также осуществление анализа поведенческих свойств без большого объема программистских усилий. Это удобно, когда новый формализм находится в стадии изучения и разработки. Были реализованы два самых изученных алгоритма анализа поведения вполне структурированных систем переходов (обратный алгоритм и анализ конечных деревьев достижимости). Их производительность была измерена на моделях сетей Петри и систем с потерей сигналов. Разработанный инструмент может быть полезным при внедрении и тестировании методов анализа формализмов, которые оказываются вполне структурированными системами переходов.

Ключевые слова: формальная верификация; системы с бесконечным числом состояний; вполне структурированные системы Переходов; сети Петри.

DOI: 10.15514/ISPRAS-2017-29(4)-11

Для цитирования: Дворянский Л.В., Михайлов В.Е. Программа поведенческого анализа вполне структурированных систем переходов. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 175-190 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-11

Список литературы

[1]. J. Burch, E. Clarke, K. McMillan, D. Dill and L. Hwang, "Symbolic model checking: 1020 States and beyond", *Information and Computation*, vol. 98, no. 2, pp. 142-170, 1992.

[2]. A. Finkel, “Well structured transition systems,” Univ. Paris-Sud, Orsay, France, Res. Rep. 365, Aug. 1987.

[3]. A. Finkel and P. Schnoebelen, "Well-structured transition systems everywhere!", *Theoretical Computer Science*, vol. 256, no. 1-2, pp. 63-92, 2001.

[4]. P. Abdulla, K. Čerāns, B. Jonsson and Y. Tsay, "Algorithmic Analysis of Programs with Well Quasi-ordered Domains", *Information and Computation*, vol. 160, no. 1-2, pp. 109-127, 2000.

[5]. R. Karp and R. Miller, "Parallel program schemata", *Journal of Computer and System Sciences*, vol. 3, no. 2, pp. 147-195, 1969.

[6]. E. Kouzmin and V. Sokolov, *Well-Structured Labeled Transition Systems,* Moscow: Fizmatlit, 2005.

[7]. J. Kruskal, "The theory of well-quasi-ordering: A frequently discovered concept", *Journal of Combinatorial Theory, Series A*, vol. 13, no. 3, pp. 297-305, 1972.

[8]. T. Parr, *The definitive ANTLR 4 reference*, Raleigh, NC and Dallas, TX: The Pragmatic Bookshelf, 2013.

[9]. L. Dickson, "Finiteness of the Odd Perfect and Primitive Abundant Numbers with n Distinct Prime Factors", *American Journal of Mathematics*, vol. 35, no. 4, pp. 413-422, 1913.

[10]. E. Dijkstra, "Hierarchical ordering of sequential processes", *Acta Informatica*, vol. 1, no. 2, pp. 115-138, 1971.

[11]. S. Akshay, B. Genest, L. Hélouët, Decidable Classes of Unbounded Petri Nets with Time and Urgency. In: F. Kordon, D. Moldt (eds) Application and Theory of Petri Nets and Concurrency. PETRI NETS 2016. *Lecture Notes in Computer Science, vol 9698*. Springer, Cham

[12]. L. W. Dworzanski, Consistent Timed Semantics for Nested Petri Nets with Restricted Urgency, in: *Formal Modeling and Analysis of Timed Systems Vol. 9884*. Switzerland : Springer International Publishing, 2016. doi Ch. 1. pp. 3-18.

[13]. J. T. Schwartz, "Set Theory as a Language for Program Specification and Programming". Courant Institute of Mathematical Sciences, New York University, 1970.

[14]. R. Dewar, "SETL and the Evolution of Programming." In From Linear Operators to Computational Biology, pp. 39-46. Springer London, 2013.

Stochastic Methods for Analysis of Complex Hardware-Software Systems

[1] *A.A. Karnov <karnov@ispras.ru>*
[2] *S.V. Zelenov <zelenov@ispras.ru>*
[1] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia.
[2] *Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Abstract. In this paper we consider Markov analysis of models of complex software and hardware systems. A Markov analysis tool can be used during verification processes of models of avionics systems. In the introduction we enumerate main advantages and disadvantages of Markov analysis. For example, with Markov analysis, unlike other approaches, such as fault tree analysis and dependency diagram analysis, it is possible to analyze models of systems that are able to recovery. The main drawback of this approach is an exponential growth of models size with number of components in analyzed system. It makes Markov analysis barely used in practice. The other important problem is to develop a new algorithm for translating a model of a system to a model suitable for Markov analysis (Markov chain), since the existing solutions have significant limitations on the architecture of analyzed systems. Next we give a brief description of the context – AADL modeling language with Error Model Annex library, MASIW framework, and also give an explanation of Markov analysis method. In a main section we suggest an algorithm for translating a system model into a Markov chain, partially solving the problem of exponential growth of Markov chain. Then follows a description of further steps, and some heuristics that allow to extremely reduce running time of the algorithm. In this paper we also consider other Markov analysis tools and their features. As a result, we suggest a Markov analysis tool that can be effectively use in practice.

Keywords: Markov analysis; system safety assessment; fault modeling; complex software-hardware system.

DOI: 10.15514/ISPRAS-2016-29(4)-12

For citation: Karnov A.A., Zelenov S.V. Stochastic Methods for Analysis of Complex Hardware-Software Systems. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 191-202. DOI: 10.15514/ISPRAS-2016-29(4)-12

1. Introduction

In this paper we consider a task related to verification of models of software and hardware systems. Such systems can be, for example, control systems for airplanes, ships, medical equipment, etc. The price of error in these systems is very high, but they are too complicated for ”manually” analysis. Therefore such systems are modeled before implementation. On the stages of design, development, and verification of the models, it is necessary to constantly investigate system safety.

At present, three main methods of system safety assessment [1] are widely used: fault tree analysis, dependency diagram analysis, and Markov analysis. Each method has its own advantages and disadvantages. In this paper, Markov analysis is considered.

Markov analysis works with a Markov chain [2] – a stochastic process, which can be represented as a directed graph with weighted edges. Vertices of Markov chain represent different states, and edges are labeled by probabilities of a transition between states. The main drawback of Markov analysis is a size of Markov chains, which increases exponentially with number of components in the system. In addition, it is necessary to develop an algorithm, that takes system model and translate it to the Markov chain. These problems make Markov analysis not so popular as the other methods, and number of tools that use Markov analysis for complex systems is relatively small. However, such approach has its advantages: Markov analysis allows to look at the entire system, to consider not only causes and probabilities of certain single failure, but also investigate how various failures affect the system in the aggregate. Also Markov analysis, unlike the other approaches, allows to analyze self-recovering systems, since return to operational state is natural for Markov chains.

Thus, the task of development the Markov analysis tool of complex hardware-software systems is quite important and relevant.

2. Context

2.1 AADL and Error Model Annex

Architecture Analysis & Design Language (AADL) [3] is a language, that widely used for describing models of real-time hardware and software systems. Its features include description of both hardware (so-called execution platform) and software components of an analyzed system, and various connections between them. The models, described in AADL, may be used for documentation, for various kinds of analysis and for code generation.

Error Model Annex [4] is an extension of AADL, that allows to simulate appearance and propagation of errors in the system. For each component, a modeller can add a description of component’s behavior states, for example, operational and failed. Transitions between system states are triggered by randomly occured error events and internal errors propagated from other components. An error propagation condition may depend on certain behavior state of the component, some error events, or error propagated from environment. Each propagated error has its own type, that allows to control what is exactly happened in the system. Also transitions between states can be defined implicitly – a state of some component may be a composite state of its subcomponents.

AADL and Error Model Annex together describe not only an architecture, but also error behavior of systems. It becomes possible to evaluate such properties of models as safety, reliability, the availability of its various states and ability to recover from them.

2.2 MASIW

MASIW [5] is an open-source framework for designing and analyzing of integrated modular avionics systems, that use AADL as a modelling language.

The project designed as plugins for Eclipse IDE, includes a variety of tools for working with AADL and Error Model Annex models. There is a big number of different analysis tools, for example, a fault tree analysis tool, but there is no Markov analysis tool.

2.3 Markov analysis

Any model subjected to Markov analysis must be represented as a Markov chain. A Markov chain can be represented in the form of a directed graph with vertices containing system states, and edges labeled with intensities of transitions between corresponding states. A Markov chain has the property of Markov process – a probability of a transition to any state depends only on a current state and a moment in time, and previous transitions are unimportant (can be characterized as memorylessness).

Markov models can be divided into models with discrete and continuous time, as well as time-homogeneous (also called stationary) and time-inhomogeneous. In time-homogeneous Markov chains, the intensities of transitions are constant, while in time-inhomogeneous Markov chains they depend on time. In time-homogeneous Markov chains, transitions occur according to the binomial (or fixed) distribution for discrete-time chains, and according to the Poisson distribution for continuous-time chains.

To determine the behavior of an analyzing system, it is necessary to specify a system of differential equations. The equations follow from the Markov chain. For all Markov processes (and a Markov chain, in particular) we have the Kolmogorov-Chapman equation [6]:

$$P^{(t+dt)}(S_j/S_i) = \sum_{k=1}^{n} P^{(dt)}(S_k/S_i)P^{(t)}(S_j/S_k) \qquad (1)$$

This equation means that probability of a transition from state $S_j$to state $S_i$for some time $t + dt$is equal to a sum of probabilities of passes into the target state $S_i$through all of intermediate states S_k.

Consider a time-homogeneous chain with an intensity of the transition between states $S_i$and $S_j$equal to $\lambda(S_i/S_j)$. Then for continuous-time Markov chains, the Kolmogorov-Chapman equation implies a system of differential equations

$$\frac{dP^{(t)}(S_j/S_i)}{dt} = -\sum_{k=1}^{n} \lambda(S_i/S_k)P^{(t)}(S_j/S_i) + \sum_{k=1}^{n} \lambda(S_k/S_i)P^{(t)}(S_j/S_k) \qquad (2)$$

And for discrete-time chains, a system of difference equations

$$\frac{P^{(t+\Delta t)}(S_j/S_i) - P^{(t)}(S_j/S_i)}{\Delta t} = -\sum_{k=1}^{n} \lambda(S_i/S_k)P^{(t)}(S_j/S_i) + \sum_{k=1}^{n} \lambda(S_k/S_i)P^{(t)}(S_j/S_k) \qquad (3)$$

Denote by S_1a certain initial state of the system, and consider equations (2)-(3) in case when $S_1 = S_j$. Denote by $P_i(t)$the function $P^{(t)}(S_1/S_i)$. Then, the previous equations takes the following form:

$$\frac{dP_i(t)}{dt} = -\sum_{k=1}^{n} \lambda(S_i/S_k)P_i(t) + \sum_{k=1}^{n} \lambda(S_k/S_i)P_k(t) \qquad (4)$$

$$\frac{P_i(t+\Delta t) - P_i(t)}{\Delta t} = -\sum_{k=1}^{n} \lambda(S_i/S_k)P_i(t) + \sum_{k=1}^{n} \lambda(S_k/S_i)P_k(t) \qquad (5)$$

In addition, initial conditions appear:

$$P_1(0) = 1, P_i(0) = 0, i = -2, n \qquad (6)$$

Thus, we obtain the Cauchy problem [7]. The solution of this problem is a set of probabilistic functions of being a system in a definite state. This is the result of Markov analysis.

In this paper, we consider only the analysis of time-homogeneous Markov chains and models, as the most common ones. However, all results can be applied to time-inhomogeneous models, with the only difference being that intensities of Markov chain transitions depend on time, and they need to be stored as formulas, not as numbers.

3. Problem

The goal of this work is a development and implementation of a Markov analysis tool for complex hardware-software systems models within the MASIW framework. The tool takes input of some system model and a set of time points. At the output, the analyzer provides the probabilities of being the system in each of its possible states at moments of time, defined by user.

The main problem is to create a Markov chain on the basis of the original model. First, we need an algorithm that creates a correct Markov chain corresponding to the input data. Secondly, the result chain should be of acceptable size, so that the program can work for acceptable time in limited memory.

After a construction of a Markov chain, further action reduces to solving a Cauchy problem with a system of linear differential equations. An analytical solution of the Cauchy problem is too complicated, resource-intensive, and result is difficult to comprehend, so we use numerical methods.

4. Solution

4.1 Markov chain

The primary task is to translate an AADL model into a Markov chain. In particular, it is necessary to find out what to regard as a chain node and what generates transitions between system states.

Obviously, the node must contain the state of the system, which is a combination of the states of all system components (the states of the components are described in the model as behavior states). However, if we take as a node any of all possible combinations, then the number of nodes will be no less than 2^n, where $n$is the number of system components. Real systems often contain more than 20 components, that, on the one hand, are few, but on the other hand, results in size of such Markov chain outside available memory.

We suggest the following solution of this problem. Let us exclude from the chain all unreachable states of the system, which, as practice shows, are the vast majority. First, some states of the system are unreachable by definition of ananalyzing model. For example, the state of some component may completely depend on the states of its subcomponents. Accordingly, the component can not be in a failed state, while all its subcomponents are in operational states. Second, the failure of some components entails an almost immediate failure of others – for example, a breakdown of a processor entails a failure of all processes running on it. Thus, the state in which the processor is broken, but the processes on it are still working, though reachable in theory, at the very moment of the failure, but instantly replaced by another state.

Thus, we suggest the following approach. We assume that speed of error propagation between components is extremely small in comparison with time of system operation (which, in practice, is the case – for a unit of time measurement usually takes hour and even a day). Let us define a stable state of the sistem as a state, that does not change until new error events occur in the system and its components. We consider as nodes of designed Markov chain only the stable states of the system. The sets of arising events generate transitions between nodes of the chain.

For the sake of saving memory, we insert only reachable states to the Markov chain, and build it dynamically, from the initial state of the system, which is a combination of the initial states of the components. In each new node it is necessary to analyze transitions from the current state of the system. The state can change for some event or combination of events. So, we perform complete search for all possible sets of events – either of them can initiate a new transition. The probability of occurrence of each set of events is easily calculated, since each event contains information about its probability distribution. This is a multiplication product of probabilities of occurrence or negation of occurrence of each of the events, since all events are independent. The total probability of all sets of events, according to the law of total probability, should be equal to 1.

The algorithm is completed when all nodes of Markov chain are analyzed, starting from the node corresponding to the initial state of the system.

```
markovChain.addNode(initialStateNode)
queue.add(initialStateNode)
while not queue.isEmpty() do
    analyzeNode(queue.head())
    queue.add(newNodes)
end while
```

The analysis of each node of Markov chain looks like this: all possible sets of error events are searched, for each of them we calculate a stable state of the system into which the given set leads, and then a new transition (and, if necessary, a new node) is added to the chain.

```
for each errorEventSet in possibleSets do
    state = currentNode.getState()
    repeat
        watchedStates.add(state)
        state = calculateState(state, errorEventSet)
    until watchedStates.contains(state)
    node = markovChain.addNode(state)
    markovChain.addTransition(
        currentNode, node, errorEventSet.getProbability())
    watchedStates.clear()
end for
```

In the above algorithm, the state of the system is considered stable if we have already reached it before. This correctly handles the case when the state of the system has not changed – we have reached the same state as in the previous step. However, in theory, in a self-recovering systems, cycling may occur if an event with a failure and an event with component recovery occur simultaneously. With this condition, the loop stops, but this situation is not handled correctly. One of the main opportunities for further improvement of the algorithm is to improve the condition for achieving a stable state of the system.

4.2 Calculation of new states

In the previous paragraph, a general algorithm for constructing a chain was described, omitting the details of calculating new states of the system. To find out exactly how the system has changed, it is enough to go through all its components, and see what transitions between states are triggered for a given set of events and the current state of the system. The triggered transition is immediately applied to the system, and the algorithm step is completed.

```
for each componentState in systemState do
    for each compositeState in comp.getCompositeStates() do
        if checkStateExpression(compositeState.getExpression())
        then
            systemState.applyTransition(compositeState)
            return
        end if
    end for
    for each transition in comp.getTransitions() do
        if transition.getSource() == compState
        and checkErrorCondition(transition.getCondition())
        then
            systemState.applyTransition(transition)
            return
        end if
    end for
end for
```

The checkStateExpression and checkErrorCondition functions check whether the transition condition is met. Such conditions can be interpreted as a logical formula, where variables corresponding to components behavior states, error events, and propagated errors, have value of true or false, depending on whether the system is in this state, whether an error event has occurred or whether an error of the specified type has propagated.

As soon as some component of the system changes its state, it means that we obtain a new state of the system, and the step of the algorithm is completed. If none of the transitions is triggered, then the system state has not changed, which is noticed by the algorithm described in the previous section.

4.3 Construction and solution of the Cauchy problem

After construction of a Markov chain, the final stage of the Markov analysis of the system is to construct a system of equations and solve the Cauchy problem. As mentioned earlier, each node of the Markov chain generates a differential equation (4) (or similar difference equation (5)). To save memory, it is not necessary to store the system of equations – the equation for any node can be easily constructed dynamically, passing through all transitions entering into this node and outgoing from it.

The resulting Cauchy problem can be solved by a numerical method from the Runge-Kutta [8] family of methods. In the analyzer, two methods are implemented: the Euler method, for discrete-time Markov chains, and the fourth-order Runge-Kutta method, for continuous-time Markov chains. The type of the chain is determined in advance, according to probability distributions of error events. An algorithm for calculating the variation of the function $P_i(t)$on each time interval delta t, taking into account the dynamic construction of the equation (Euler's method):

```
for each node in markovChain.getNodes() do
    i = indexOf(node)
    res = 0
    for each transition in node.getInTransitions() do
        k = indexOf(transition.getNode())
        res += transition.getProbability() * pPrev[k]
    end for
    for each transition in node.getOurTransitions() do
        res -= transition.getProbability() * pPrev[i]
    end for
    pCur[i] = pPrev[i] + delta t * res
end for
```

Also, the value of the vector of probability functions P (t) is saved at every time point defined by user. As soon as values at each necessary time point are calculated, the algorithm is completed.

4.4 Getting Analysis Results

Since number of system states in Markov chain can be very large, the result of analysis in the form of probabilities of being the system in each of them is practically impossible for reading. Considering that each system has its root component, we group all system states according to the states of the root component.

In this case, all the probability functions within the same group are summed up:

```
for each node in chainNodes do
    i = indexOf(node)
    state = node.getSystemState().get(rootComp)
    analysisResult.get(state) += p[i]
end for
```

After this, for each state of the root component, the probability of being the system in a this state at given time points is obtained. This is the desired result of the Markov analysis of the system.

4.5 Algorithm acceleration

Despite a partial solution of the problem of exponential growth of Markov chain size, the running time of full version of the algorithm still grows exponentially — due to a thorough search of all possible combinations of error events. Thus, we use some heuristics in the final program, accelerating the algorithm.

First, we limit the search of combinations of events. Since the probability of occurrence of one event is usually extremely small, the situation in which several events occur simultaneously is practically impossible. Therefore, a very small numerical parameter, limiting the probability of the combination of events under consideration, was added to the program. If the probability of occurrence of the set of events is less than this parameter, then the effect of the set of events on the system is not considered. This solution significantly reduced the running time of the program, without much loss of accuracy of the result.

The second solution relates to system's ability to self-recovery. In practice, there are few examples of self-recovering systems, and, in most cases, even a short-term failure of the system itself means fatal consequences. Accordingly, if the analyzed system has come to failed state, its further changes are not interesting to us — no matter what else can fail in the already failed system. Therefore, we introduce a set of states of the root component, that are considered as «absolutely» failed. If some node of Markov chain has failed state of the root component, then we do not analyze transitions from it. If analyzed system is not self-recovering, the result of the program remains the same, but is obtained in much shorter time.

Both modifications of the program are optional, as they may change final result in some cases, but their application reduces the operating time by several orders of magnitude. For example, a complete analysis of a system containing 24 components revealed 919 states of the Markov chain and took 1 hour. Limiting the frequency of the events considered by the number 10^{-30}gave a significant gain — the same set of states of the Markov chain and the same result of the analysis were obtained in 7 minutes. Since the system under test was not self-recovering system, the analysis with the stop-on-failed option was correct, and got the same result in 10 seconds. Setting relevant parameters allows to significantly accelerate work of the analyzer. One of the further options for improving the tool can be automatic detection and selection of optimizing parameters.

5. Related works

Markov analysis of AADL and Error Model Annex models is usually applied to systems consisting of only one component. Such algorithms doesnt consider error propagation mechanism and composite states, and limited by root component.

The tool from OSATE [9] framework, created for export AADL model into Markov chain model for PRISM [10] toolset, which provide further steps of Markov analysis, supports only the first nesting level of the component hierarchyand does not support different types of propagated errors. In addition, there were some problems associated with the syntactic correctness of the final PRISM model.

6. Conclusion

In this paper we present a new Markov analysis tool, and in particular, an algorithm for translating AADL and Error Model Annex models into Markov chains. In addition, there were added some improvement for accelerating the algorithm, which make it possible to effectively use the tool in practice.

The presented tool can be further improved in various ways: adding support for time-inhomogeneous Markov chains, accelerating the work of the algorithm, changing some details of algorithm.

References

[1]. "SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," Warrendale, USA, Dec. 1996.

[2]. A. N. Shiryaev, Probability (2Nd Ed.). Secaucus, NJ, USA: Springer Verlag New York, Inc., 1995.

[3]. P. H. Feiler and D. P. Gluch, Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language, 1st ed. Addison-Wesley Professional, 2012.

[4]. P. Feiler, "SAE AADL error model annex: An overview," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep., 2014. [Online]. Available: https://wiki.sei.cmu.edu/aadl/images/1/13/ErrorModelOverview-Sept222011-phf.pdf

[5]. "MASIW framework," https://forge.ispras.ru/projects/masiw-oss/.

[6]. S. Kuznetsov, "Mathematical models of processes and systems of technical exploitation of avionics as Markov and semi-Markov processes," Civil Aviation High Technologies [Nauchnyi Vestnik MGTU GA], no. 213, pp. 28–33, 2015 (in Russian).

[7]. J. Hadamard, Lectures on Cauchy's Problem in Linear Partial Differential Equations (Dover Phoenix Editions). Dover Publications, 2003.

[8]. U. M. Ascher and L. R. Petzold, Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations. SIAM: Society for Industrial and Applied Mathematics, 1998.

[9]. J. Delange, P. Feiler, D. Gluch, and J. Hudak, "AADL fault modeling and analysis within an ARP4761 safety assessment," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2014-TR-020, 2014. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=311884

[10]. M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in Proc. 23rd International Conference on Computer Aided Verification (CAV'11), ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

Стохастические методы анализа комплексных программно-аппаратных систем

[1] А.А. Карнов <karnov@ispras.ru>
[2] С.В. Зеленов <zelenov@ispras.ru>
[1] Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1.
[2] Институт системного программирования РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В данной работе рассматривается марковский анализ моделей комплексных программно-аппаратных систем. Инструмент марковского анализа может быть использован, в частности, для верификации моделей систем интегральной модульной авионики. Во введении перечисляются основные достоинства и недостатки марковского анализа. К примеру, марковский анализ, в отличие от других подходов — анализа дерева неисправности и анализ алогической схемы, позволяет анализировать модели систем, способных к восстановлению. Основным недостатком данного подхода является экспоненциальный рост размера моделей в зависимости от числа компонентов в анализируемой системе. Это существенно ограничивает возможность применения марковского анализа на практике. Другой важной проблемой является создание нового алгоритма трансляции исходной модели системы в модель, пригодную для марковского анализа (марковскую цепь), так как существующие решения накладывают существенные ограничения на архитектуру анализируемой системы. Далее идет краткое описание контекста, в котором инструмент должен работать — язык моделирования AADL с библиотекой Error Model Annex, набор инструментов MASIW, а также описывается сам метод марковского анализа. В основной части предлагается алгоритм трансляции модели системы в марковскую цепь, частично решающий проблему экспоненциального роста марковской цепи. Затем следует описание дальнейших шагов, а также предлагаются эвристики, позволяющие значительно сократить время работы итоговой программы. В работе также рассматриваются существующие инструменты марковского анализа и их недостатки. В качестве результата данной работы предлагается новый инструмент марковского анализа, который может быть эффективно использован на практике.

Ключевые слова: Марковский анализ; оценка безопасности системы; моделирование неисправностей; комплексные программно-аппаратные системы.

DOI: 10.15514/ISPRAS-2016-29(4)-12

Для цитирования: Карнов А.А., Зеленов С.В. Стохастические методы анализа комплексных программно-аппаратных систем. Труды ИСП РАН, том 29, вып. 4, 2017 г., стр. 191-202. DOI: 10.15514/ISPRAS-2016-29(4)-12

Список литературы

[1]. “SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment,” Warrendale, USA, Dec. 1996.

[2]. Ширяев А.Н. *Вероятность*. Москва: Наука, 1989.

[3]. P. H. Feiler and D. P. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language, 1st ed*. Addison-Wesley Professional, 2012.

[4]. P. Feiler, “SAE AADL error model annex: An overview,” Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep., 2014. [Online]. Available: https://wiki.sei.cmu.edu/aadl/images/1/13/ErrorModelOverview-Sept222011-phf.pdf

[5]. “MASIW framework,” https://forge.ispras.ru/projects/masiw-oss/.

[6]. Кузнецов, С.В. “Математические модели процессов и систем технической эксплуатации авионики как марковские и полумарковские процессы,” Научный Вестник МГТУ ГА, 2015, No 213, стр. 28-33.

[7]. J. Hadamard, *Lectures on Cauchy’s Problem in Linear Partial Differential Equations (Dover Phoenix Editions)*. Dover Publications, 2003.

[8]. U. M. Ascher and L. R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM: Society for Industrial and Applied Mathematics, 1998.

[9]. J. Delange, P. Feiler, D. Gluch, and J. Hudak, “AADL fault modeling and analysis within an ARP4761 safety assessment,” Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2014-TR-020, 2014. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=311884

[10]. M. Kwiatkowska, G. Norman, and D. Parker, “PRISM 4.0: Verification of probabilistic real-time systems,” in Proc. 23rd International Conference on Computer Aided Verification (CAV’11), ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

Predicate Abstractions Memory Modeling Method with Separation into Disjoint Regions

[1] *A. Volkov <arvolkov@inbox.ru>*
[2] *M. Mandrykin <mandrykin@ispras.ru>*
[1] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia
[2] *Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Abstract. Software verification is a type of activity focused on software quality control and detection of errors in software. Static verification is verification without the execution of software source code. Special software – tools for static verification – often work with program's source code. One of the tools that can be used for static verification is a tool called CPAchecker. The problem of the current memory model used by the tool is that if a function returning a pointer to program's memory lacks a body, arbitrary assumptions can be made about this function return value in the process of verification. Although possible, the assumptions are often also practically very improbable. Their usage may lead to a false alarm. In this paper we give an overview of the approach capable of resolving this issue and its formal specification in terms of path formulas based on the uninterpreted functions used by the tool for memory modeling. We also present results of benchmarking the corresponding implementation against existing memory model.

Keywords: memory model; predicate abstractions; static verification

DOI: 10.15514/ISPRAS-2017-29(4)-13

For citation: Volkov A., Mandrykin M. Predicate Abstractions Memory Modeling Method with Separation into Disjoint Regions. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 203-216. DOI: 10.15514/ISPRAS-2017-29(4)-13

1. Introduction

Software verification is a type of activity focused on software quality control and detection of errors in software [1]. Static verification is a verification without the execution of software source code.

Special software – tools for static verification – often work with program's source code. Depending on the tools used for static verification it is possible to conduct analysis of the source code to search for errors in program's behavior.

One of the tools that can be used for static verification is a tool called CPAchecker. It takes program's source code as an input, creates a CFA (control-flow automaton) and uses it to run the analysis. One of the analyses the instrument is capable of is a reachability analysis. In this paper we consider reachability properties that can be expressed as checking if the call to an error function is reachable. Its strong side is that the CPA (configurable program analysis) [2] concept allows to use a composition of several analyses for program verification. The tandem of Value Analysis and Predicate Analysis produces good results in terms of verification precision / verification time ratio.

2. Definitions and notations

We will call a model of program's memory or just a memory model a strategy of organization and representation of program's memory. By region we will refer to the set of lvalues with the following restriction: if two lvalues are taken from two different regions they necessarily reference disjoint memory locations [3]. For example, different regions may be safely assigned to the lvalues referring distinct structure fields under the following conditions:

- the fields do not occur as an argument to the address taking operator (&);
- the fields do not become targets of some pointers by the usage of pointer type conversion or address arithmetic.

The situation when a program's error state is reachable due to the imprecisions of abstraction employed in the analysis is called a false alarm.

3. CPAchecker's memory model

Existing memory model employed by Predicate Analysis of the CPAchecker tool uses uninterpreted functions. Each of those functions has only a name and a number of arguments. If $f(x)$ is an uninterpreted function, a and b are any of its arguments for which $a = b$ is true then $f(a) = f(b)$[4]. Uninterpreted functions in the CPAchecker tool are used to establish a correspondence between a memory location and the value stored at this memory location. Depending on the type of the expression different uninterpreted functions should be used.

Existing memory model of the CPAchecker tool uses typed regions. This means that all lvalues of the same type exist in the same region. However, a large number of lvalues of the same type is present in any big enough program written in the C programming language. This leads to the addition of a big number of logical constraints for each event of a pointer's memory update. The constraints express checks for potential equality of the updated lvalue to each memory location in the region. Those checks allow to determine precisely what memory should also be updated but noticeably increase the length of path formulas.

The problem of the current memory model used by the tool is that if a function returning a pointer to program's memory lacks a body, arbitrary assumptions can be made about its return value in the process of verification. In other words, it is considered possible for this pointer to point at any lvalue in the region. Although possible, this situation is also practically very improbable. In those cases it is hard to determine if a path leading to an error label really does or doesn't exist. One of the approaches capable of resolving this issue suggests the introduction of smaller regions that divide a bigger typed region.

4. B&B memory model

4.1 Memory model overview

B&B memory model was proposed by Richard Bornat and had been based on the work of Rod Burstall [5], [6]. It is used in Frama-C verification tool in Jessie plugin which is capable of performing verification of the C programs. In its foundation are assumptions that can introduce regions of smaller sizes instead of having very big one for a type. These assumptions state that if struct data type fields never occur as arguments to the address taking operator (&) in program's source code then those fields can be placed to separate regions. Otherwise they must belong to the same region as the normal pointers of the same type.

This memory model has some flaws. It does not take into account that the struct fields can be accessed through address arithmetic and pointer conversions. It also needs mentioning that some overhead costs are required for region support. Taking into account the pros and cons of the model it is possible to say that the B&B memory model looks promising.

4.2 Formal specification

For ease of specification we will assume the following:

- variables can only be of struct s * types;
- struct s fields can only be of int type;
- struct s has *n* fields: struct s { int f1, f2, …, fn; };

Program's memory location can be represented by an lvalue expression like pointer dereference. To model changes to the program's state when assignments to lvalues arise the CPAchecker tool uses uninterpreted functions [4].

We assume absence of pointer arithmetic and restrict pointer dereferences to the applications of the arrow operator ($p \rightarrow f_i$), where p is a pointer to the struct type and f_i is one of the struct fields).

Let ϒ be a set of uninterpreted functions. It consists of the uninterpreted function *G* that is used for accessing a memory location in global region, a finite number of uninterpreted functions F_i, where each function F_i represents the state of the memory region corresponding to lvalues of the form $b \rightarrow f_i, i = \overline{1,n}$ and the uninterpreted function *undef_ptr* with zero arity that models the usage of the program's functions returning an unknown pointer.

Let $B(e)$ be an uninterpreted function used for global memory location modeling and $B_i(e)$, $i = \overline{1,n}$ — a finite set of uninterpreted functions used for memory location modeling in regions corresponding to F_i uninterpreted functions. For address representation it is suggested to use expressions like a, where a is a variable. The axioms of the memory model (positivity of addresses and their non-intersection within one region) can be represented as follows:

- $a > 0$;
- $B(a) = k$, where k is a unique number for each such variable.

The tool uses SSA representation to model the varying state of program variables and memory regions. In this representation usage of a name splits into usages of its versions. Each time an assignment happens to a program variable or a memory region represented by the corresponding variable or uninterpreted function in the path formula, the version number (index) of that variable or an uninterpreted function increases.

Let *Index* : $\Upsilon \rightarrow \mathbb{N}$ be a mapping of a set of uninterpreted functions Υ to a numerical set of their indices.

Let $Alloc\colon \Upsilon \rightarrow Addrs$ be a mapping of a set of uninterpreted functions Υ to the set of subsets of memory locations $Addr$: $Addrs = 2^{Addr}$.

We will use a supplementary function *mem_upd*:

$$mem\_upd(p, f, m', m) = \bigwedge_{a \in Alloc(f)} ((p = a) \vee (f_{m'}(a) = f_m(a)))$$

that defines a check for address equality for all of the lvalues in the same region as pointer p (locations in the *Alloc*(*f*) region are modeled by the uninterpreted function f, m = *Index*(f) is a current version of f and $m' = m + 1$ is a new version).

We define $\omega(s, f_i)$ as a constant offset of a field f_i from the base address of struct type variable s. Because we assume that there is only one structure type struct s in our programs, $\omega(s, f_i)$ can be made just $\omega(f_i)$.

In B&B memory model implemented on top of CPAchecker's existing memory model the operator of a strongest post-condition is defined as $SP(op(\phi)) = \phi \wedge \Gamma(op)$, where ϕ is a symbolic abstract state and constraints $\Gamma(op)$ are defined by table 1.

4.3 Example

The following program will be considered correct if we use either of the memory models. Γ constraints in terms of B&B memory model for the program are shown in table 2. Path formula can be made as a conjunction of all formulas in Γ column of the table 2. It is *unsat* in terms of either of the memory models. This means that the tool cannot go by this path (i.e. won't consider it as a potential error trace candidate).

```
struct s { int f1, f2; };
struct s * p1;
struct s * p2;
p1 = alloc();
p2 = alloc();
p1 -> f1 = 6;
p2 -> f2 = 5;
assume(p1 -> f1 == p2 -> f2);
```

Table 1. Γ constraints creation rules

Operation (op)	$Index$	$Alloc$	Base address index k'	Γ constraints
Variable allocation on stack struct s * p;	No changes	*A'* - new variable, *Alloc'(G)* = *A'* ∪ *Alloc(G)*	k' - new index	*p = A' ∧ A' > 0 ∧ B(A') = k'*
Heap variable allocation p = alloc()	l' - new index for G, *l = Index(G)*, *Index' = Index* \ {*G*→*l*}∪{*G*→*l'*}	A', A'_i - new variables, *Alloc'(G)* = *A* ∪ *Alloc(G)* $Alloc'(F^i) = A'_i \cup Alloc(F^i), i = \overline{1, n}$	k', k'_i - new indices, $i = \overline{1, n}$	$G_{l'}(p) = A' \wedge A' > 0 \wedge B(A') = k'$ ∧ *mem_upd*(*p*,*G*,*l*,*l'*) $\bigwedge_{i=\overline{1,n}}((G_{l'}(p)+\omega(f_i)) = A'_i \wedge A'_i > 0 \wedge B^i(A'_i) = k'_i)$
p=*undef_ptr*()	l' - new index for G, *l = Index*(*G*), m' - new index for *undef_ptr*, *m* = *Index*(*undef_ptr*), *Index' = Index* \ ({*G* → *l*}∪ {*undef_ptr* →*m*})∪	No changes	No changes	$G_{l'}(p) = undef\_ptr_m \wedge$ *mem_upd*(*p*,*G*,*l*,*l'*)

	$\{G \rightarrow l'\} \cup$ $\{undef\_ptr \rightarrow m'\}$			
p→ f_i = e	m' - new index for F^i, m= $Index(F^i)$, $Index' = Index \setminus \{F^i \rightarrow m\} \cup \{F^i \rightarrow m'\}$	No changes	No changes	$F^i_{m'}(G_l(p) + \omega(f_i)) = \Gamma(e) \land mem\_upd(G_l(p) + \omega(f_i), F^i, m', m)$, where *l = Index(G)* and $\Gamma(e)$ can be computed using the following rules: Γ(*const*) : *const;* $\Gamma(p2 \rightarrow f_j)$: $F^j_k(G_l(p2) + \omega(f_j))$, where *k*=*Index*($F^j$), *l* = *Index*(*G*); $\Gamma(e_1$ *op* $e_2)$, *op* ∈ {‘+’, ‘−’, ‘*’, ‘\’}: $\Gamma(e_1)$ *op* $\Gamma(e_2)$.
assume(p)	No changes	No changes	No changes	$\Gamma(p)$ for predicate *p* can be computed as following: Γ(*const*) : *const*; $\Gamma(s)$: G_l(s), where *l* = *Index*(*G*); $\Gamma(s \rightarrow f_i)$: $F^i_m(G_l(s) + \omega(f_i))$, where *m* = *Index*($F^i$), *l* = *Index*(*G*); $\Gamma(p1 == p2) : \Gamma(p1) == \Gamma(p2)$; $\Gamma(p1 < p2) : \Gamma(p1) < \Gamma(p2)$; $\Gamma(p1 <= p2) : \Gamma(p1) \leq \Gamma(p2)$; $\Gamma(p1 \parallel p2) : \Gamma(p1) \lor \Gamma(p2)$; $\Gamma(p1\ \&\&\ p2) : \Gamma(p1) \land \Gamma(p2)$; $\Gamma(!p) : \neg\Gamma(p)$.

Table 2. Example build of path formula for the correct program

Path instruction	*Index*	*Alloc*	k'	Γ
struct s * p1;	{G→1, F^1 →1, F^2 →1}	$Alloc$(G)={A_1}	1	$p1 = A_1 \wedge A_1 > 0 \wedge B(A_1) = 1$
struct s * p2;	{G→1, F^1 →1, F^2 →1}	$Alloc$(G)={A_1, A_2}	2	$p2 = A_2 \wedge A_2 > 0 \wedge B(A_2) = 2$
p1 = alloc();	{G→2, F^1 →1, F^2 →1}	$Alloc$(G)={A_1, A_2, A_3} $Alloc(F^1)$ = {A_4} $Alloc(F^2)$ = {A_5}	3,4,5	$G_2(p1) = A_3 \wedge A_3 > 0 \wedge B(A_3) = 3$ $\wedge (G_2(p1)+\omega(f_1)) = A_4 \wedge A_4 > 0 \wedge B(A_4) = 4$ $\wedge (G_2(p1)+\omega(f_2)) = A_5 \wedge A_5 > 0 \wedge B(A_5) = 5$
p2 = alloc();	{G→3, F^1 →1, F^2 →1}	$Alloc$(G)={A_1, A_2, A_3,A_6} $Alloc(F^1)$ = {A_4, A_7} $Alloc(F^2)$ = {A_5,A_8}	6,7,8	$G_3(p2) = A_6 \wedge A_6 > 0 \wedge B(A_6) = 6$ $\wedge (G_3(p2)+\omega(f_1)) = A_7 \wedge A_7 > 0 \wedge B(A_7) = 7$ $\wedge (G_3(p2)+\omega(f_2)) = A_8 \wedge A_8 > 0 \wedge B(A_8) = 8$
p1→f1 = 6;	{G→3, F^1 →2, F^2 →1}	$Alloc$(G)={A_1, A_2, A_3,A_6} $Alloc(F^1)$ = {A_4, A_7} $Alloc(F^2)$ = {A_5,A_8}	8	$F^1_2(G_3(p1)+\omega(f_1)) = 6$ $\wedge$ $mem\_upd(G_3(p1)+\omega(f_1), F^1,2,1)$
p2→f2 = 5;	{G→3, F^1 →2, F^2 →2}	$Alloc$(G)={A_1, A_2, A_3,A_6} $Alloc(F^1)$ = {A_4, A_7} $Alloc(F^2)$ = {A_5,A_8}	8	$F^2_2(G_3(p2)+\omega(f_2)) = 6$ $\wedge$ $mem\_upd(G_3(p2)+\omega(f_2), F^2,2,1)$
assume(p1→f1 == p2→f2)	{G→3, F^1 →2, F^2 →2}	$Alloc$(G)={A_1, A_2, A_3,A_6} $Alloc(F^1)$ = {A_4, A_7} $Alloc(F^2)$ = {A_5,A_8}	8	$F^1_2(G_3(p1)+\omega(f_1)) = F^2_2(G_3(p2)+\omega(f_2))$

Why the conjunction is *unsat*?

1) In the existing memory model memory allocated for pointers $p1$ and $p2$ cannot intersect because it was allocated using the known $alloc()$ function (the corresponding path formula is not given).

2) In the given Γ constraints for this path (using the B&B model) the following contradicting elements are present:
   - $F_2^1(G_3(p1) + \omega(f1)) = F_2^2(G_3(p2) + \omega(f2))$;
   - $F_2^1(G_3(p1) + \omega(f1)) = 5$;
   - $F_2^2(G_3(p2) + \omega(f2)) = 6$.

Let's take a look at the example program below. In the program's source code there are calls to the function *undef_ptr*() that returns an unknown pointer. The pointer $p2$ is initialized using this function. Γ constraints in terms of B&B memory model for the program are shown in table 3. Path formula can be made as conjunction of all formulas in Γ column of the table 3.

```
void * undef_ptr();
struct s { int f1, f2; };
struct s * p1;
struct s * p2;
p1 = alloc();
p2 = undef_ptr();
p1 -> f1 = 6;
p2 -> f2 = 5;
assume(p1 -> f1 == p2 -> f2);
```

In B&B memory model $p1 \rightarrow f1$ and $p2 \rightarrow f2$ exist in the separate memory regions. In Γ constraints for this path the same contradicting elements as for the previous example are present. Thus, the update of one of them wouldn't affect the other one. Because of that the result of verification would be that the error state is unreachable (path formula is still *unsat*).

However, in the existing memory model fields $f1$ and $f2$ of struct s exist in the same memory region and it uses only one uninterpreted function for them (see table 2 in [4]). Memory for their base pointers $p1$ and $p2$ was allocated using known *alloc*() function and function *undef_ptr*() returning unknown pointer respectively. It cannot be confirmed that an update to a field $f2$ of the $p2$ wouldn't affect the access to the $f1$ struct field of $p1$. In the formula the location for field $f2$ of the $p2$ is $(G_3(p2) + \omega(f2))$ which is $undef\_ptr_1 + \omega(f2)$. Locations $(G_3(p1) + \omega(f1))$ and $(G_3(p2) + \omega(f2))$ exist in the same region and may be equal. Thus the formula is satisfiable. It means that the result of verification with existing memory model will be a reachable path to the program's error state.

Usually such situations in practice are false alarms because different fields of different structures do not normally intersect. Thus, the assumptions related to this behavior in the existing memory model aren't really incorrect but they are quite improbable in practice. Usage of the B&B memory model will be able to reduce the number of false alarms caused by these assumptions (continued in section 6).

Table 3. Example build of path formula for the program with unknown memory function

Path instruction	*Index*	*Alloc*	k'	Γ
struct s * p1;	{G→1, F^1 →1, F^2 →1, *undef_ptr*→1}	*Alloc*(G) ={A_1}	1	$p1 = A_1 \wedge A_1 > 0 \wedge B(A_1) = 1$
struct s * p2;	{G→1, F^1 →1, F^2 →1, *undef_ptr*→1}	*Alloc*(G) ={A_1, A_2}	2	$p2 = A_2 \wedge A_2 > 0 \wedge B(A_2) = 2$
p1 = alloc();	{G→2, F^1 →1, F^2 →1, *undef_ptr*→1}	*Alloc*(G) ={A_1, A_2, A_3} *Alloc*(F^1) = {A_4} *Alloc*(F^2) = {A_5}	3,4,5	$G_2(p1) = A_3 \wedge A_3 > 0 \wedge B(A_3) = 3$ $\wedge (G_2(p1)+\omega(f_1)) = A_4 \wedge A_4 > 0 \wedge B(A_4) = 4$ $\wedge (G_2(p1)+\omega(f_2)) = A_5 \wedge A_5 > 0 \wedge B(A_5) = 5$
p2=*undef_ptr*();	{G→3, F^1 →1, F^2 →1, *undef_ptr*→2}	*Alloc*(G) ={A_1, A_2, A_3} *Alloc*(F^1) = {A_4} *Alloc*(F^2) = {A_5}	5	$G_3(p2) = undef\_ptr_1$ ∧ *mem_upd*(*p*2,*G*,3,2)
p1→f1 = 6;	{G→3, F^1 →2, F^2 →1, *undef_ptr*→2}	*Alloc*(G) ={A_1, A_2, A_3} *Alloc*(F^1) = {A_4} *Alloc*(F^2) = {A_5}	5	$F^1_2(G_3(p1)+\omega(f_1)) = 6$ ∧ *mem_upd*($G_3(p1)+\omega(f_1)$, F^1,2,1)
p2→f2 = 5;	{G→3, F^1 →2, F^2 →2, *undef_ptr*→2}	*Alloc*(G) ={A_1, A_2, A_3} *Alloc*(F^1) = {A_4} *Alloc*(F^2) = {A_5}	5	$F^2_2(G_3(p2)+\omega(f_2)) = 6$ ∧ *mem_upd*($G_3(p2)+\omega(f_2)$, F^2,2,1)
assume(p1→f1 == p2→f2)	{G→3, F^1 →2, F^2 →2, *undef_ptr*→2}	*Alloc*(G) ={A_1, A_2, A_3} *Alloc*(F^1) = {A_4} *Alloc*(F^2) = {A_5}	5	$F^1_2(G_3(p1)+\omega(f_1)) = F^2_2(G_3(p2)+\omega(f_2))$

5. Implementation notes

The creation of memory regions is an automated process. In CPAchecker verification tool CFA (control-flow automaton) is used as an inner representation of the program. It is sufficient to go through it and find in it all of the struct field accesses. This allows to distinguish those fields that don't have their address taken somewhere in the program.

In the implementation we do not take into consideration the possibility of field accesses through pointer arithmetic and through the usage of pointer conversions because of the high improbability of such field accesses in program's source code.

6. Experiments

To determine the efficiency of B&B memory model implementation in comparison to existing memory model of the CPAchecker tool a number of launches were performed on the predefined sets of Linux kernel modules. To use the implemented memory model one must have:

- CPAchecker verification tool with revision number 23271 or higher from the branch trunk;
- option cpa.predicate.useMemoryRegions should be set to 'true'.
- The following experiments were made using the revision *trunk:*23271 of the tool.

6.1 False alarm set

The review of error traces obtained during the verification of Linux kernel 3.14 allowed to determine situations when reachability of error state was present due to updates to same-typed pointers' memory. This set consists of those 26 kernel modules that caused false alarms due to the updates to pointer's memory. The goal of this experiment was to find out what effect the usage of B&B memory model will have on the tools precision. Tables 4 and 5 hold information about changes of the tool's verdicts.

Table 4. B&B applicability

	B&B could help	B&B could not help
B&B helped	10	0
B&B did not help	0	16

Table 5. Verdict changes

False alarm → Safe	False alarm → Unsafe	False alarm → False alarm*
3	5	2

* - different error trace and cause of Unsafe

6.2 Linux 4.2-rc1 kernel modules

A set of Linux kernel drives (version 4.2-rc1) was selected to study the efficiency of B&B memory model implementation in comparison to the existing memory model of the CPAchecker tool.

The launch was performed for rule that checks correctness of functions working with `usb_get_*` and `usb_put_*` functions of `usb-system`. Launch results can be found in tables 6, 7.

Launch configuration:

- time limit – 15 minutes;
- memory limit – 15 Gb;
- number of CPU cores – 4;

The differences in the regions the models have led to the difference in program's paths that are covered by the tool. This explains Unsafe → Unknown, Unknown → Safe and Unknown → Unsafe transitions, where Safe means that program's error state is unreachable, Unsafe – error state is reachable, Unknown – timeout or runtime error. This experiment's results show that the improvement to the tool's precision is present while the verification speed remains competitive.

Table 6. Linux 4.2-rc1 statistics

	Existing model	B&B
Verification time	35.8 hours	35.3 hours
Safe	4245	4241
Unsafe	69	68
Unknown	161	166

Table 7. Transitions

Existing model \ B&B model	Safe	Unsafe	Unknown
Safe	4240	0	5
Unsafe	0	67	2
Unknown	1	1	159

6.3 SV-COMP'17 DeviceDrivers64

This set contains files from the DeviceDrivers64 set of the international competition on software verification SV-COMP'17. It consists of 2795 modules of different Linux kernel versions. Launch results can be found in tables 8, 9, 10.

Launch configuration:

- time limit – 15 minutes;
- memory limit – 15 Gb;

- number of CPU cores – 4;

Several of the transitions from the incorrect results can be explained by the difference in models' choice of pointer's may-aliases. The same modules were present in the earlier mentioned False alarm set. Several transitions to Unknown can be explained by the additional overhead costs required for B&B usage to the verification tasks on the verge of timeout.

Table 8. DeviceDrivers64 statistics

Memory models	Existing	B&B
Total number of files	2795	2795
Correct results	1791	1780
Error state unreachable	1524	1522
Error state reachable	267	258
Incorrect results	7	5
Missed errors	4	4
False alarms	3	1
Unknown	997	1010

Table 9. Time for DeviceDrivers64 set

Memory models	Existing	B&B
Total time	143.6 hours	143.1 hours
Time for correct results	14.9 hours	14.1 hours
SMT solver time	10500 sec (2.9 hours)	12400 sec (3.4 hours)
SMT solver time for correct results	660 sec	605 sec

Table 10. Transitions

Existing model \ B&B model	Correct results	Incorrect results	Unknown
Correct results	1775	0	16
Incorrect results	2	5	0
Unknown	3	0	994

7. Conclusion

This paper proposes the specification of B&B memory model and its region-based reasoning in terms of uninterpreted functions. Its implementation on top of existing memory model of the CPAchecker verification tool provides better verification precision while the verification speed remains competitive. The implementation was included in the official repository of the CPAchecker static verification tool.

References

1. V. Kuliamin, “Software verification methods,” Vserossiiskii konkursnyi otbor obzornoanaliticheskikh statei po prioritetnomu napravleniyu ”Informatsionnotelekommunikatsionnye sistemy” [Russian national competitive selection of review and analytical articles in priority direction ”Information and telecommunication systems”], 117 p., 2008 (in Russian).
2. D. Beyer, T. A. Henzinger, and G. Theoduloz, “Configurable Software Verification: Concretizing the Convergence of Model Checking and Program Analysis,” in Computer Aided Verification, ser. Lecture Notes in Computer Science, W. Damm and H. Hermanns, Eds., vol. 4590. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 504–518.
3. M. Mandrykin and A. Khoroshilov, “A memory model for deductively verifying Linux kernel modules,” A.P. Ershov Informatics Conference, the PSI Conference Series, 11th edition, 2017 (to appear).
4. M. Mandrykin and V. Mutilin, “Modeling Memory with Uninterpreted Functions for Predicate Abstractions,” Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 5, 2015, pp. 117–142 (in Russian). DOI: 10.15514/ISPRAS-2015-27(5)-7
5. R. Bornat, “Proving pointer programs in Hoare Logic,” in Mathematics of Program Construction: 5th International Conference, MPC 2000, ser. Lecture Notes in Computer Science, R. Backhouse and J. N. Oliveira, Eds., vol. 1837. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 102–126.
6. R. Burstall, “Some techniques for proving correctness of programs which alter data structures,” Machine Intelligence, vol. 7, pp. 23–50, 1972.

Метод моделирования памяти в предикатных абстракциях с разделением на непересекающиеся области

[1] *А.Р. Волков <arvolkov@inbox.ru>*
[2] *М.У. Мандрыкин <mandrykin@ispras.ru>*
[1] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1.*
[2] *Институт системного программирования РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25.*

Аннотация. Верификация программного обеспечения — вид деятельности, направленный на контроль качества программного обеспечения и обнаружения ошибок в нем. Статическая верификация - это один из способов верификации, который производится без выполнения исходного кода программы. Для статической верификации используется специальное программное обеспечение: инструменты статической верификации, которые часто работают с исходным кодом программы. Одним из таких инструментов является инструмент под названием CPAchecker. Проблема его текущей модели памяти заключается в том, что при встрече функции, возвращающей указатель на область памяти, у которой отсутствует тело, в процессе верификации о ее возвращаемом значении могут быть сделаны произвольные предположения. Несмотря на то, что они теоретически допустимы, вероятность их выполнения на практике очень низка. Использование этих предположений может привести к ложному предупреждению в качестве результата верификации. В данной статье мы делаем обзор на один из подходов, благодаря которому можно избавиться от такой проблемы, а также предлагаем формальное описание данного подхода в терминах формул путей, содержащих неинтерпретируемые функции, которые инструмент использует для моделирования памяти программы. Также мы приводим результаты сравнительного анализа эффективности предложенной реализации относительно существующей модели памяти.

Ключевые слова: модель памяти; предикатные абстракции; статическая верификация.

DOI: 10.15514/ISPRAS-2017-29(4)-13

Для цитирования: Волков А.Р., Мандрыкин М.У. Метод моделирования памяти в предикаты абстракциях с разделением на непересекающиеся области. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 203-216 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-13

Список литературы

1. В.В. Кулямин, “Методы верификации программного обеспечения” Всероссийский конкурсный отбор обзорно-аналитических статей по приоритетному направлению "Информационно-телекоммуникационные системы", 117 стр., 2008.
2. D. Beyer, T. A. Henzinger, and G. Theoduloz, “Configurable Software Verification: Concretizing the Convergence of Model Checking and Program Analysis,” in Computer Aided Verification, ser. Lecture Notes in Computer Science, W. Damm and H. Hermanns, Eds., vol. 4590. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 504–518.
3. M. Mandrykin and A. Khoroshilov, “A memory model for deductively verifying Linux kernel modules,” A.P. Ershov Informatics Conference, the PSI Conference Series, 11th edition, 2017 (to appear).
4. Мандрыкин М.У. и Мутилин В.С., “Моделирование памяти с использованием неинтерпретируемых функций в предикатных абстракциях” Труды ИСП РАН, том 27, вып. 5, 2015, стр. 117–142. DOI: 10.15514/ISPRAS-2015-27(5)-7
5. R. Bornat, “Proving pointer programs in Hoare Logic,” in Mathematics of Program Construction: 5th International Conference, MPC 2000, ser. Lecture Notes in Computer Science, R. Backhouse and J. N. Oliveira, Eds., vol. 1837. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 102–126.
6. R. Burstall, “Some techniques for proving correctness of programs which alter data structures,” Machine Intelligence, vol. 7, pp. 23–50, 1972.

Static Verification of Linux Kernel Configurations

[1] *S.V. Kozin <kozyyy@yandex.ru>*
[2] *V.S. Mutilin <mutilin@ispras.ru>*
[1] *National Research University Higher School of Economics,*
20 Myasnitskaya Ulitsa, Moscow, 101000, Russia
[2] *Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. The Linux kernel is often used as a real world case study to demonstrate novel software product line engineering research methods. It is one of the most sophisticated programs nowadays. To provide the most safe experience of building of Linux product line variants it is necessary to analyse Kconfig file as well as source code. Ten of thousands of variable statements and options even by the standards of modern software development. Verification researchers offered lots of solutions for this problem. Standard procedures of code verification are not acceptable here due to time of execution and coverage of all configurations. We offer to check the operating system with special wrapper for tools analyzing built code and configuration file connected with coverage metric. Such a bundle is able to provide efficient tool for calculating all valid configurations for predetermined set of code and Kconfig. Metric can be used for improving existing analysis tools as well as decision of choice the right configuration. Our main goal is to contribute to a better understanding of possible defects and offer fast and safe solution to improve the validity of evaluations based on Linux. This solution will be described as a program with instruction for inner architecture implementation.

Keywords: Software Product Lines, Linux, Kconfig, Preprocessor

DOI: 10.15514/ISPRAS-2017-29(4)-14

For citation: Kozin S.V., Mutilin V.S. Static Verification of Linux Kernel Configurations. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 217-230. DOI: 10.15514/ISPRAS-2017-29(4)-14

1. Introduction

Nowadays, software is used to solve increasingly important and complex tasks, due to this fact the complexity of software architectures is also constantly growing. With the increasing complexity of programs, the complexity of development, analysis and maintenance arises. There are many methods that allow you to reduce the costs of supporting the software life cycle. One such method is the creation of variable systems (or family of systems, software product families, software product lines). The superiority over the usual development method is that systems are manufactured with the condition of multiple elements used for several systems with a similar set of functions, taking into account a specific target audience of users. At the same time, a complex and widely known representation of variable software is the Linux operating system [1-5].

In the development of variable systems, a variability model and a variation mechanism play a fundamental role. The variability model specifies the space of possible variants of this family of systems. Usually it is determined by a set of features or configuration parameters, by the sets of their possible values and constraints on possible combinations of these values, each variant of the system corresponds to a certain set of values of all features. The variation mechanism provides the ability to build all possible system variants from a limited set of created and followed artifacts. In Linux, the variability model and its relationship to the variation mechanism is built on the basis of Kconfig files, Makefile files and additional scripts. Kconfig describes all possible features, as well as their relationship with each other in a special language. Then on the basis of Kconfig, the configuration file .config is defined, which describes the version of the system. It consists of a set of configuration variables described in Kconfig and values that satisfy the constraints of Kconfig. During kernel assembly, the values of variables specified in .config are passed to the code as constants for the preprocessor and to Makefiles, which will be used in the variation mechanism. Makefiles contain information about objects in kernel: what files are included and which mode of compilation will be used [5].

In the field of operating systems (hereinafter the operating system we mean the kernel and the underlying OS libraries, providing the interfaces for work with computing resources and hardware), a mechanism of conditional compilation of C / C ++ languages is widely used as a mechanism for variability of the mixed type (based on macros #ifdef, #if, else #else). Blocks that are surrounded with variability mechanism macros, are called variable blocks. It allows you to compose code at the build stage that combines various variables specified by a set of characteristic values that are conditional compilation parameters in this case (defined by the #define and #undef macros, as well as preprocessor setup parameters). The expression after macros is called block precondition, if configuration turns it into ‘true’, block gets compiled [5].

The complexity of variability models for modern operating systems is very high, for example, the Linux kernel version 2.6.32 has 6319 characteristics, more than 10,000 constraints that can be used up to 22 individual characteristics, with the majority of characteristics depending on at least 4 others, and the maximum depth of the dependency tree is 8 [6]. This complexity causes a large number of errors, primarily due to the difficulty of taking into account all the factors that a developer of a separate code element should do. To identify and cope with these errors, it is necessary to use specialized techniques of analysis and verification. Complexity of analysis that is typical for systems with such a variability mechanism arise because of the huge size of the possible variants space (which makes it completely unrealistic to check them all). Due to using of conditional compilation, each fragment does not have to be a separate component with a certain behavior that can be analyzed separately from the rest of the code, usually such fragments are just insertions into the common code, and can only be checked in certain combinations with each other. The need to solve these problems imposes special requirements on the tools and analysis methods used for complex variable operating systems and system software in general. These requirements are specific for analysis and verification - the methods used to create such systems, by themselves, do not facilitate their analysis [7, 8]. The main goal of this work is to propose a method capable of coping with the verification of the Linux OS taking into account the variability, to give acceptable accuracy of verification and speed of execution comparable to the verification of common programs.

An errors analysis of such complex systems as Linux can be done with a lot of different tools. The most convenient of them are static analyzers and static verifiers. Static code analysis is the analysis of software produced (as opposed to dynamic analysis) without real execution of the programs under investigation. Existing static analyzers (such as Coverity [9] or Svace [10]) and static verifiers (such as BLAST, CPAchecker [11]) are only designed to work with the code already compiled. Accurate analysis requires pre-assembling of a specific configuration, and only after that start of the actual analysis. As a result, the total inspection time becomes unacceptably large. There is a class of tools that are focused on analysis of a set of possible configurations, they do not split the phase of building configurations and code analysis. As the example of such tools we can take a look at TypeChef and Undertaker. These tools are designed to solve special problems in the sphere of variability. Undertaker is looking for a "dead code" - such a problems when different configurations give the same product, besides it has a lot of built-in helper modules to provide main task, and one more function - assembling the minimal Linux kernels for individual use cases. TypeChef is looking for linking and compile errors with a variability-aware method. It is important to say, that TypeChef can not find difficult problems in code like complex static analysers. Suggested in this paper tool should analyze code deeply like BLAST or CPAchecker and, on the other hand, should do it in variability-aware way without all-configuration brute forcing. For this task we suggest to use CPAchecker due to its outstanding abilities in the error findings, despite CPAchecker’s expenditure of time [11]. The maximum reduction in verification time should be achieved through the optimal selection of configurations that will be directed to the input of the static analyzer.

2. Configuration Set Selection

The problem of selecting configurations can be considered as a splitting of the configuration space into equivalence classes. Each of the selected configurations will belong to one of the classes. To split the configuration space into classes, it is suggested to use the MC/DC test coverage criterion. The advantage of this choice is that it allows you to significantly reduce the number of classes, even for large software systems. In addition, methods of analysis/construction of configurations for preprocessor directives are similar to methods for test data generation for coverage of programs in programming language, where the MC/DC criterion has proven itself well. If we take a look at standard usage of MC/DC for code coverage, we can find out that conditionals in usual programming language has the same structure as conditional compilation directives, the difference is in compilation (conditional compiling directives will not compile code under the block if condition is not true, which is the same as not giving control to code inside usual conditionals.

There are 2 basic notions in MC/DC: *decision* and *condition*. *Decision* is a propositional formula which consists of conditions. If it is true, then control will be given to block with such decision (in the case of preprocessor - code will be compiled). Otherwise, control will not be given to this block (code will not be compiled). *Condition* is a logical part of decision which connects to other conditions with logical functions.

A set is considered to reach 100% coverage by this metric, if:

1. Each decision takes every possible outcome.
2. Each condition in a decision takes every possible outcome.
3. Each condition in a decision is shown to independently affect the outcome of the decision.

In other words, in the full test, in accordance with the MC/DC coverage criterion, it should be demonstrated that every condition that can influence the resulting value of the decision that includes it actually changes its value regardless of the other conditions.

Example:

Some module of Linux has variable block inside. For example, consider a decision: A && B || C; where A, B, C are some boolean constraints of Kconfig.

The decision is applicable for variable block (Fig. 1).

```
#if (A && B || C)
   …
#endif
```

Fig. 1. Example of variable block

We can extract the conditions out of the decision: A, B, C.

That means that we have to build such table for this variable block (Table 1).

Table 1. Truth table for variable block in Fig. 1

A	B	C	Decision
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Now we will find those pairs of conditions values where the change of one of them affects on decision. For each condition we have to get only one pair, and then choose minimal amount of them to cover all conditions. This will satisfy third MC/DC point. Pairs will be (Table 2)

Table 2. MC/DC coverage table

A	B	C	Decision	A	B	C
0	0	0	0			C1
0	0	1	1			C1
0	1	0	0	A1		C2
0	1	1	1			C2
1	0	0	0		B1	C3
1	0	1	1			C3
1	1	0	1	A1	B1	
1	1	1	1			

According to table we will test only pairs that are marked as A1, B1 and C3 or C2; These 2 are minimal sets for MC/DC coverage.

We can notice that from 8 possible combinations, we can use only 4 to get full coverage according to MC/DC method (0-1-0, 0-1-1,1-0-0,1-1-0).

In general, the MC / DC metric allows 2n different situations to be used instead of n^2 condition combinations.[14]

3. Kernel Check Stages

For each of the found configurations, you should run a kernel verification. The program will scan the kernel in 6 stages: configuration, search for a "dead code", preprocessing, compiling, linking, searching for run-time errors. It is also worth noting that we will look for not only errors, but also configuration defects. Defects - is a broader concept, and it includes not only system errors, but also possible errors of the kernel without processing the interrupt.

Description of the stages:

A. Search for a “dead code”.
A "dead" code is such a code in which control is not transferred under any circumstances. This code contributes to a common configuration error when two different configurations produce the same product at the output. This problem can be solved using the built-in tools of the Undertaker.

B. Configurations.
At the time of build, Linux itself checks the configuration file, but it's worth checking it for recursive dependencies and non-existent variables in the code, but existing in Kconfig (and vice versa).

C. Preprocessing.
Preprocessing is performed just before compilation and at this stage we will get the code that will be compiled into the final version of the program. Most of errors at this stage can find by a preprocessor. It remains for us to inform the user of a possible conflict of names if we see duplicate names of preprocessor variables and their redefinition, because they have one nominal space and the developer may not notice the problem of overriding.

D. Compilation.
Compilation is complete on the compiler side. Here there are such errors as: detection of an undeclared variable / function, missing punctuation in the code, etc.

E. Linking.
As well as compilation is completely redirected to the linker. The linker finds errors in missing libraries or files.

F. Execution Errors.
The most difficult part of the test is using the LDV and CPAchecker tools. LDV assigns labels to the code of the program according to the preset rules, while CPAchecker searches for them and builds accessibility graphs to them so we can see how a label is achieved.

4. Configuration Set Selection

To solve this task we suppose the workflow for a program called OStap (Fig. 2). This program will find all variable blocks, get all the propositional formulas for entering each of the variable blocks. After that program will extract all conditions from those formulas and will use **MC/DC** metric to get all necessary configurations that will be checked with static verifier and dead code detector to perform verification stages A-F.

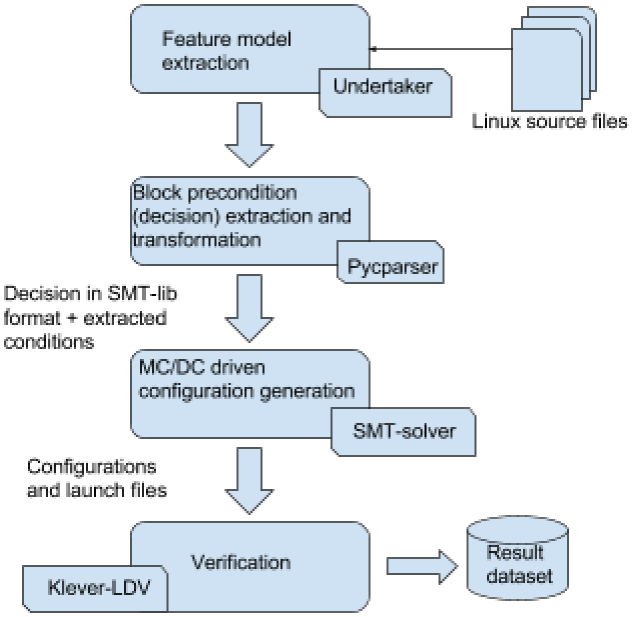

Feature model extraction
Undertaker
Linux source files
Block precondition (decision) extraction and transformation
Pycparser
Decision in SMT-lib format + extracted conditions
MC/DC driven configuration generation
SMT-solver
Configurations and launch files
Verification
Klever-LDV
Result dataset

Fig. 2. OStap workflow.

4.1 Feature model extraction

First of all, we transform Kconfig files using Kconfigdump module of **Undertaker** to get feature model of Linux kernel. Feature model is splitted by architectures sets of formulas. All dependencies implemented in Kconfig file are represented in a dump as a set of logical formulas. So it is possible to get full precondition for each feature just looking at line with it's name in model file. For example: if we have to turn on option A to turn on option B there will be …&&A addition to any decision where config B is have to be turned on.

4.2 Block precondition extraction

4.2.1. Coarse block precondition extraction

Secondary, we need to get all variable blocks in the given code, which is a module or a set of modules that consists of different .c files. These modules can be compiled according to Makefiles, where modules are marked as compiled, compiled as LKM, non-compiled. This mechanism of variability (Kbuild variability) in Linux is made for modules and helps to organize builds. Due to the fact, that another variability mechanism in Linux uses preprocessors we can find all #ifdef, #if, #else, #ifndef blocks to be sure that we have found all the variable blocks in the given code. This may be done using standard tools of most of popular operating systems or using programming language tools for work with file system.

When we have all necessary blocks and their positions in the files we can get propositional formulas using **Undertaker**[13] tool for them. As a result of the undertaker infrastructure we can easily calculate preconditions for preprocessor blocks in a file, just by specifying the file and line number. If a configuration model is loaded, it will also fetch all interesting items from the model. Say you want to have the **block precondition** for line 359 and line 370 in init/main.c.

Then for each block we use **Undertaker's** option -blockpc to get blocks precondition based on dumped feature model and code. This precondition is a decision in **MC/DC** metric theory. It is also necessary to say that later we will use **SMT-solver** to work with **MC/DC** metrics, so we patched Undertaker for providing output in a SMT-lib way (Polish notation). C expressions with operands, are not able to be converted due to architecture of undertaker, such code may be marked with special symbols to be changed in the future.

```
$ undertaker –j blockpc init/main.c:359
init/main.c:370
I: Block B20 | Defect: no | Global: 0
B20
&&
( B18 <->  ! CONFIG_SMP )
&& ( B20 <->  ( B18 )  && CONFIG_X86_LOCAL_APIC )
&& ( B22 <->  ( B18 )  && ( ! (B20) )  )
&& ( B25 <-> ( ! (B18) )  )

I: Block B25 | Defect: no | Global: 0
B25
&&
( B18 <->  ! CONFIG_SMP )
&& ( B20 <->  ( B18 )  && CONFIG_X86_LOCAL_APIC )
&& ( B22 <->  ( B18 )  && ( ! (B20) )  )
&& ( B25 <-> ( ! (B18) )  )
```

Fig. 3. Example of usual Undertaker output without model.

```
$ undertaker -j blockpc -m
models/x86.model  init/main.c:370
I: loaded rsf model for x86
I: Using x86 as primary model
I: Block B25 | Defect: no | Global: 0
B25
&&
( B18 <->  ! CONFIG_SMP )
&& ( B20 <->  ( B18 )  && CONFIG_X86_LOCAL_APIC )
&& ( B22 <->  ( B18 )  && ( ! (B20) )  )
&& ( B25 <-> ( ! (B18) )  )

&&
(CONFIG_X86_32 -> ((!CONFIG_64BIT)))
&&
(CONFIG_X86_32_NON_STANDARD -> ((CONFIG_X86_32 &&
CONFIG_SMP && CONFIG_X86_EXTENDED_PLATFORM)))
&&
(CONFIG_X86_64 -> ((CONFIG_64BIT)))
&&
(CONFIG_X86_EXTENDED_PLATFORM -> ((CONFIG_X86_64)
&& (CONFIG_X86_32)))
&&
(CONFIG_X86_LOCAL_APIC -> ((CONFIG_X86_64 ||
CONFIG_SMP || CONFIG_X86_32_NON_STANDARD ||
CONFIG_X86_UP_APIC) && (CONFIG_X86_64 ||
CONFIG_SMP || CONFIG_X86_32_NON_STANDARD ||
CONFIG_X86_UP_APIC)))
&&
(CONFIG_X86_UP_APIC -> ((CONFIG_X86_32 &&
!CONFIG_SMP && !CONFIG_X86_32_NON_STANDARD)))
```

Fig. 4. Example of usual Undertaker output with model.

4.2.2 Detailed block precondition extraction

When we have formula by **Undertaker**, we have to change all the marked code to prefix view, in other words: to represent decision in SMT-lib way. Due to the fact that such code will be in a usual C representation, we can use any **C parser** that is able to build expression tree to rebuild the string. For example we can use **Pycparser** [15]. **Pycparser** can parse C code and represent it as ast-tree. Ast-tree is an expression tree with all operators of C language. Running through the tree, we can rewrite any C expression from infix to prefix view. To do this thing we need to apply this algorithm:

```
Algorithm prefix (tree)
    if (tree not empty)
          print (tree token)
          prefix (tree left subtree)
          prefix (tree right subtree)
    end if
end prefix
```

4.3 MC/DC driven configuration generation

Now we have ready-to-use prefix formula that is the same as ‘decision’ term in **MC/DC** metrics so it is time to start extracting conditions from a decision. To pass this stage we need to get all the single variables inside unary operator or without unary operator. Those variables will be the same as conditions.

When we have all conditions and decision for a variable block, we need to build a truth table for conditions. For example, we extracted 3 conditions: a > 0, b == true, a < 0, and out decision is (a > 0) && (b == true) || (a < 0). We look over all of their combinations that we got from truth table like it was described in II chapter (Table 3).

Table 3. Truth table without decision for (a > 0) && (b == true) || (a < 0).

Line	A > 0	B = true	A < 0	decision
1	0	0	0	
2	0	0	1	
3	0	1	0	
4	0	1	1	
5	1	0	0	
6	1	0	1	
7	1	1	0	
8	1	1	1	

Each line of the table reflects set of equalities related to column and value. For our example line 1 means: a > 0 = false, b == true = false, a < 0 == false, whereas line 6 means: a > 0 = true, b == true = false, a < 0 == true;

To calculate decision column we need to put such equalities and full decision formula into **SMT-solver**, so it calculates possible values of variables through equalities and then put it into decision formula to find solution. In our example in case of a > 0 = true and a < 0 = true **SMT-solver** will return error, so these sets of variables will not be used in future (Table 4).

Table 4. Truth table with decision for (a > 0) && (b == true) || (a < 0).

Line	A > 0	B = true	A < 0	decision
1	0	0	0	0
2	0	0	1	1
3	0	1	0	0
4	0	1	1	1
5	1	0	0	0
6	1	0	1	Error
7	1	1	0	1
8	1	1	1	Error

In classical MC/DC theory we have to say that this example can not be covered by MC/DC due to the fact that 6 and 8 line gave us impossible conditions to resolve, but in a real case adaptation we can say that we covered all of possible conditions [14]. When the table is ready, we can find influencing variables, like it was described in chapter II (Table 5).

Table 5. MC/DC coverage table for (a > 0) && (b == true) || (a < 0).

Line	A > 0	B = true	A < 0	decision	A > 0	B ==true	A < 0
1	0	0	0	0			1
2	0	0	1	1			1
3	0	1	0	0			2
4	0	1	1	1			2
5	1	0	0	0		3	
6	1	0	1	Error			
7	1	1	0	1		3	
8	1	1	1	Error			

For MC/DC coverage we need A/B and C pairs to check. That means that we have to extract a and b values related to those lines. These values form configurations that we will check with help of verifier. Each line for each configuration.

4.4 Verification

Finally, we have got necessary configurations to get 100% coverage using **MC/DC** metrics. Now we will push these configurations to static verifier (for example, **CPAchecker**). Also we will check code with **Undertaker** tool to find dead code blocks (Stage A of kernel check), which is also can be declared as configuration error.

Next step for pushing configurations is to create launch file for **LDV-Klever** tool and launch it. Launch file is filled with modules-to-check and rules for finding unsafe modules [16]. Before the launch, code will be compiled and stages B-E will be passed with compiler default methods. After that **LDV-Klever** main activity will be invoked to check code for a runtime defects (stage F). Output is a module with result: safe, unsafe or unknown. If module is unsafe, verifier shows error trace for this module.

During verifier launches we push results into the dataset where the structure unite modules, that we are checking, each module is splitted with configurations and results of **LDV-Klever**, **LDV-Klever** results are splitted with verdict and unsafe traces.

5. Conclusion

This article describes the method, which allows you to check the Linux operating system for errors without being depended on the configuration. This approach provides high code coverage and an improved speed of verification comparing to brute force method.

The prototype of this program is already implemented. It will be finished and fixed in a nearest time.

In future work MC/DC will be better adapted to current aim(Linux kernel), also project will be tested in a real work during ISP RAN researches. There will be also replacement in used tools, probably or rewriting them. This software product can be used in the production of distributions, as well as for verification of existing ones. As a result we will get linear depended amount of configurations for full testing of Linux systems.

References

[1]. Jacobson I., Griss M., Jonsson P. Software Reuse, Architecture, Process and Organization for Business Success. Addison-Wesley, 1997.

[2]. Bosch J. Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach. Pearson Education, 2000.

[3]. Clements P., Northrop L. Software Product Lines: Practices and Patterns. SEI Series in Software Engineering, Addison-Wesley, 2001.

[4]. Pohl K., Böckle G., van der Linden F. J. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, 2005. DOI: 10.1007/3-540-28901-1.

[5]. Kuliamin V.V., Lavrischeva E.M., Mutilin V.S., Petrenko A.K. Verification and analysis of variable operating systems. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 3, 2016, pp. 189-208 (in Russian). DOI: 10.15514/ISPRAS-2016-28(3)-12

[6]. Lotufo R., She S., Berger T., Czarnecki K., Wąsowski A. Evolution of the Linux kernel variability model. Proc. of SPLC'10, LNCS 6287:136-150, Springer, 2010. DOI: 10.1007/978-3-642-15579-6_10.

[7]. Lavrischeva E. M., Koval G.I., Slabospickaya O.O., Kolesnik A.L. Features of management processes when creating families of software systems [Osobennosti processov upravleniya pri sozdanii semejstv programmnyh system]. Problems of programming [Problemy programmirovaniya], 3:40-49, 2009 (in Russian).

[8]. Lavrischeva E.M., Koval G.I., Slabospickaya O.O., Kolesnik A.L. Teoreticheskie aspekty upravleniya variabel'nost'yu v semejstvah programmnyh sistem. Bulletin of KSU, a series of physics and mathematics [Vesnik KNU, seriya fiz.–mat. nauk], 1:151-158, 2011 (in Russian).

[9]. Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, Dawson Engler, A "Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World", Communications of the ACM, Vol. 53 No. 2, pp. 66-75

[10]. Borodin A.E., Belevancev A.A, A Static Analysis Tool Svace as a Collection of Analyzers with Various Complexity Levels, Trudy ISP RAN/Proc. ISP RAS, vol 27, issue. 6, 2015, pp. 111-134. DOI: 10.15514/ISPRAS-2015-27(6)-8 (in Russian).

[11]. Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, "The software model checker BLAST", Int J Softw Tools Technol Transfer (2007) 9:505–525, Springer-Verlag 2007

[12]. Andy Kenner, Christian Kastner, SteffenHaase, Thomas Leich, "TypeChef: toward type checking #ifdef variability in C". Proceeding FOSD '10 Proceedings of the 2nd Internationsal Workchop on Feature-Oriented Software Development, pp. 25-32, Eindhoven, The Netherlands, Oct. 10, 2010.

[13]. Stephan Henglein. Vampyr configurability aware compile testing of source files. Linux Plumber Conference, Oct 15-17, 2014, Dusseldorf, Germany. Available at:

http://www.linuxplumbersconf.net/2014/ocw//system/presentations/2313/original/hengelein.pdf, accessed 12.01.2017.

[14]. Kulyamin V., Model-based testing [Testirovanie na osnove modeley]. (online publication). Available at: http://mbt-course.narod.ru/Lecture03.pdf, accessed 12.02.2017 (in Russian).

[15]. Alber Zever. Pycparcer wiki. (Onine publication). Available at: https://pypi.python.org/pypi/pycparser/2.14. accessed 7.05.2017.

[16]. I.S. Zaharov, M.U. Mandrykin, V.S. Mutilin, E.M. Novikov, A.K. Petrenko, A.V. Khoroshilov. Configurable Toolset for Static Verification of Operating Systems Kernel Modules. Trudy ISP RAN/Proc. ISP RAS, vol 26, issue 2, 2014, pp. 5-42 (in Russian). DOI: 10.15514/ISPRAS-2014-26(2)-1.

Статическая верификация конфигураций ядра Linux

[1] *С.В. Козин <kozyyy@yandex.ru>*
[2] *В.С. Мутилин <mutilin@ispras.ru>*
[1] *Национальный исследовательский университет Высшая Школа Экономики, 101000, Россия, г. Москва, ул. Мясницкая, д. 20.*
[2] *Институт системного программирования РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25.*

Аннотация. Ядро операционной системы Linux – это частый пример современных инженерных решений в области создания продуктовых линеек программного обеспечения. Сегодня это одна из наиболее сложных программных систем. Для того, чтобы обеспечить наиболее безопасное построение вариантов продуктовой линейки, необходимо анализировать конфигурационный файл Kconfig помимо исходного кода. Ядро содержит десять тысяч вариабельных переменных несмотря на современную инженерию. Исследователи в области верификации предлагают большое количество решения проблемы анализа. Стандартные процедуры верификации здесь не могут быть применены из-за времени проверки покрытия всех конфигураций. Мы предлагаем инструмент, который базируется на связи уже существующих программах для проверки кода и конфигурационного файла с метрикой покрытия. Такой пакет – это эффективный инструмент для расчета всех допустимых конфигураций для предопределенного набора кода и Kconfig. Предложенные методы могут быть использованы для улучшения существующих инструментов анализа, а также для выбора правильной конфигурации. Наша основная цель – лучше разобраться в возможных дефектах и предложить быстрое и безопасное решение для проверки ядра Linux. Это решение будет описано как программа с инструкцией по реализации внутренней архитектуры.

Ключевые слова: линейка программных продуктов, Linux, Kconfig, препроцессор

DOI: 10.15514/ISPRAS-2017-29(4)-14

Для цитирования: Козин С.В., Мутилин В.С. Статическая верификация конфигураций ядра Linux. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 217-230 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-14

Список литературы

[1]. Jacobson I., Griss M., Jonsson P. Software Reuse, Architecture, Process and Organization for Business Success. Addison-Wesley, 1997.

[2]. Bosch J. Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach. Pearson Education, 2000.

[3]. Clements P., Northrop L. Software Product Lines: Practices and Patterns. SEI Series in Software Engineering, Addison-Wesley, 2001.

[4]. Pohl K., Böckle G., van der Linden F. J. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, 2005. DOI: 10.1007/3-540-28901-1.

[5]. В.В. Кулямин, Е.М. Лаврищева, В.С. Мутилин, А.К. Петренко. “Верификация и анализ вариабельных операционных систем” Труды ИСП РАН, том 28, вып. 3, 2016, стр. 189-208. DOI: 10.15514/ISPRAS-2016-28(3)-12

[6]. Lotufo R., She S., Berger T., Czarnecki K., Wąsowski A. Evolution of the Linux kernel variability model. Proc. of SPLC’10, LNCS 6287:136-150, Springer, 2010. DOI: 10.1007/978-3-642-15579-6_10.

[7]. Лаврищева К.М., Коваль Г.И., Слабоспицкая О.О., Колесник А.Л. Особенности процессов управления при создании семейств программных систем. Проблемы программирования, 3:40-49, 2009.

[8]. Лаврищева К.М., Слабоспицкий А.А., Коваль Г.И., Колесник А.А. Теоретические аспекты управления вариабельностью в семействах программных систем Теоретические аспекты управления вариабельностью в семействах программных систем. Вестник КНУ, серия физ.–мат. наук, 1:151-158, 2011.

[9]. Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, Dawson Engler, A “Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World”, Communications of the ACM, Vol. 53 No. 2, pp. 66-75

[10]. Бородин А.Е., Белеванцев А.А., “Статический анализатор Svace как коллекция анализаторов разных уровней сложности” Труды ИСП РАН, том 27, вып. 6, 2015, стр. 111-134. DOI: 10.15514/ISPRAS-2015-27(6)-8

[11]. Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, “The software model checker BLAST”, Int J Softw Tools Technol Transfer (2007) 9:505–525, Springer-Verlag 2007

[12]. Andy Kenner, Christian Kastner, SteffenHaase, Thomas Leich, “TypeChef: toward type checking #ifdef variability in C”. Proceeding FOSD ’10 Proceedings of the 2nd Internationsal Workchop on Feature-Oriented Software Development, pp. 25-32, Eindhoven, The Netherlands, Oct. 10, 2010.

[13]. Stephan Henglein. Vampyr configurability aware compile testing of source files. Linux Plumber Conference, Oct 15-17, 2014, Dusseldorf, Germany. Available at: http://www.linuxplumbersconf.net/2014/ocw//system/presentations/2313/original/hengelein.pdf, дата обращения 12.01.2017.

[14]. Кулямин В., Тестирование на основе моделей. (online publication). http://mbt-course.narod.ru/Lecture03.pdf, дата обращения 12.02.2017.

[15]. Alber Zever. Pycparcer wiki. (Onine publication). Available at: https://pypi.python.org/pypi/pycparser/2.14. дата обращения 7.05.2017.

[16]. И.С. Захаров, М.У. Мандрыкин, В.С. Мутилин, Е.М. Новиков, А.К. Петренко, А.В. Хорошилов. Конфигурируемая система статической верификации модулей ядра операционных систем. Труды ИСП РАН, том 26, вып. 2, 2014, стр. 5-42. DOI: 10.15514/ISPRAS-2014-26(2)-1.

A Technique for Parameterized Verification of Cache Coherence Protocols

V.S. Burenkov <burenkov_v@mcst.ru>
JSC MCST, 24 Vavilov str., Moscow, 119334, Russian Federation

Abstract. This paper introduces a technique for scalable functional verification of cache coherence protocols that is based on the verification method, which was previously developed by the author. Scalability means that verification efforts do not depend on the model size (that is, the number of processors in the system under verification). The article presents an approach to the development of formal Promela models of cache coherence protocols and shows examples taken from the Elbrus-4C protocol model. The resulting formal models consist of language constructs that directly reflect the way protocol designers describe their developments. The paper describes the development of the tool, which is written in the C++ language with the Boost.Spirit library as parser generator. The tool automatically performs the syntactical transformations of Promela models. These transformations are part of the verification method. The procedure for refinement of the transformed models is presented. The refinement procedure is supposed to be used to eliminate spurious error messages. Finally, the overall verification technique is described. The technique has been successfully applied to verification of the MOSI protocol implemented in the Elbrus computer systems. Experimental results show that computer memory requirements for parameterized verification are negligible and the amount of manual work needed is acceptable.

Keywords: multicore microprocessors; shared memory multiprocessors; cache coherence protocols; model checking; Spin; Promela.

DOI: 10.15514/ISPRAS-2017-29(4)-15

For citation: Burenkov V.S. A Technique for Parameterized Verification of Cache Coherence Protocols. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 231-246. DOI: 10.15514/ISPRAS-2017-29(4)-15

1. Introduction

Shared memory multiprocessors constitute one of the most common classes of high-performance computer systems. In particular, multicore microprocessors, which combine several processors (cores) on a chip, are widely used [1]. The number of cores is constantly increasing. The presence of cache memories that are local to each core determines the need for ensuring coherent memory state. To satisfy the need, microprocessor developers design and implement in hardware cache coherence protocols [2].

Cache coherence mechanisms are extremely complex. Therefore, both the design and their implementation are error-prone. Being especially critical, protocol bugs should be revealed before implementing the hardware. The widely recognized method for protocol verification is *model checking* [3]. It is fully automated, but suffers from a principal drawback – it is not scalable due to the *state space explosion* problem. Verification of a cache coherence protocol for five or more processors is impossible (at least, highly problematic) with the traditional methods [4].

To overcome the problem and develop scalable verification technologies, researchers focus mostly on verification of *parameterized designs* [3]. Previous articles of the author [5–8] presented a method for parameterized verification of cache coherence protocols. The author successfully applied the method to verification of the cache coherence protocol of the Elbrus-4C computing system. This paper presents an approach to the development of formal Promela models that can be analyzed by the verification method, describes the development of the tool that performs transformations of Promela models according to the method and presents the overall verification technique.

The paper is structured as follows. Section 2 takes a brief look at related work and provide the necessary links. Section 3 considers the question development of Promela models of cache coherence protocols. In Section 4, we describe how to perform parameterized verification of the Promela models in a semi-automatic way. We examine the development of the tool that automates parts of the verification method used. We present a technique for cache coherence protocols verification. Section 5 provides experimental results on using the technique for verifying the Elbrus-4C protocol. Section 6 summarizes the work and defines further research directions.

2. Related Work

This work extends the previous works [5–8] by dealing with the question of practical application of the method for parameterized verification of cache coherence protocols presented in those works.

Article [5] presents a review of related work and gives the motivation for development of a new method. The developed method is based upon works [9–13] that present a method of compositional model checking, which is based on syntactical transformations of models written in the Murφ language and counterexample-guided abstraction refinement.

The method [5–8] is used in the context of the following verification process:

1) Development of formal models of cache coherence protocols.

2) Parameterized verification by means of the method.

3. Development of Formal Models

It is highly desirable to have a modeling language that allows us to conveniently describe cache coherence protocols. To choose or develop such a language, we need to define a mathematical model of cache coherence protocols.

In accordance with the microprocessor system model that is used in work [2] for representation and analysis of cache coherence protocols, I chose to model cache coherence protocols as a set of communicating finite-state machines.

An element of this set may be either a cache controller or the system commutator. Let us define these notions. Each memory device of the microprocessor is operated by a *coherence controller*, which is a finite-state machine. Coherence controllers are coordinated by a special device – the *system commutator* – that is also a finite-state machine. A set of these machines constitutes a distributed system, in which the machines communicate by message passing in order to maintain cache coherence.

Each coherence controller connected with cache memory logically implements a set of independent and identical finite-state machines, one for each cache line. These machines are called *cache controllers*. Due to the independence and identity of cache controllers, it is customary to reflect only one cache line in the models of cache coherence protocols.

The states of cache controllers are divided into two classes: Stable states and transient states. Stable states of cache controllers are often the subset of the common set Modified, Owned, Exclusive, Shared, Invalid [2]. Transitions between these states are not atomic and occur through transient states. Transient states are specific to each microprocessor and their presence is one of the factors that determine high verification complexity.

Conditions that define correctness of cache coherence protocols are formulated as statements about stable states, for example: “Cache line can never be in Modified state in two caches simultaneously” [5]. Such statements belong to the class of invariant properties [14].

Usage of a set of communicating finite-state machines as the model of cache coherence protocols and invariant properties for specification defined the choice of the Promela language for modeling cache coherence protocols:

- In contrast to other languages (for example, Murφ and NuSMV), Promela provides process types and the means of synchronous and asynchronous interprocess communication (channels).
- Promela provides convenient specification language, which is Linear Temporal Logic (LTL).
- Spin – the system that implements Promela – provides different verification algorithms and optimizations, and is a modern and constantly developing tool.

The question of development of formal models of cache coherence protocols is insufficiently covered in the literature. Here, I present an approach to the construction of such models. According to the approach, a formal model of a cache coherence protocol of a system with n cores consists of n Promela processes for cache controllers and one Promela process for the system commutator.

For the considered cache coherence protocols, the following property holds: Only one initial request may be in process at a given point in time. System commutator performs a sequence of steps during the request processing, for example, the reception of the initial request and its analysis, sending of snoop- and other requests according to the results of the analysis, reception of the answers to these requests. Initial requests correspond to the memory access instructions that the processor core is executing. Reception of messages from other devices can only occur at particular steps. Thus, it is convenient to represent the system commutator as a Promela process whose body simply consists of operators that follow each other (Fig. 1).

```
proctype system_commutator() {
again:
  <receive initial request>
  <analyze the initial request>
  <send coherent requests>
  <receive answers to coherent requests or the request completion message>
  <finalize the request processing>
goto again }
```

Fig. 1. Structure of the System Commutator Process.

Cache controllers operate differently. On the one hand, we still may identify a number of steps, for example, sending an initial request, changing state from stable to transient, receiving snoop-requests. On the other hand, the relative order of these steps is often unspecified, and the same messages from other devices may be processed in different states of a cache controller. Thus, it is convenient to represent processes of this kind as infinite do-cycles consisting of the guarded commands (Fig. 2).

```
proctype cache_controller() {
 do
 :: <send initial request from main states>
 :: <receive and process snoop-requests>
 :: <receive answers to coherent requests>
 :: <send the completion message>
od }
```

Fig. 2. Structure of Cache Controller Processes.

See papers [5, 6, 8] for more details on how to organize processes and their communication.

For example, modeling of a situation in which cache controller sends an initial request and the system commutator receives it, may be performed as follows:

```
mtype cache[N] = I; // states of cache line
proctype cache_controller(byte i) {
do
:: atomic {cache[i] == I ->
// send initial request and change state
if :: ini_req_chan ! R, i; cache[i] = WR;
   :: ini_req_chan ! RI, i; cache[i] = WRI;
      ...
fi }
...
od }


proctype system_commutator(byte i) {
message_t message;
again:
// receive initial request
atomic {ini_req_chan ? message;
           curr_command = message.opcode;
           curr_client = message.requester;
}
if :: atomic {
// send snoop-request as a response
// to the initial request
curr_command == R ->
    coh_req_chan[0] ! snR, curr_client; ...
}
...
// receive acknowledgement
final_ack_chan ? message;
goto again; }
```

As another example, reception of a snoop-request by cache controller and generation of the response can be modeled as follows:

```
proctype cache_controller(byte i) {
do
...
:: atomic {nempty(coh_req_chan[i]) ->
    // receive snoop-request
    coh_req_chan[i] ? message;
if ...
   // analyze state...
:: cache[i] == WI_O
  // ... and the snoop-request type
  && message.opcode == snI ->
    // send corresponding answer
    coh_ans_chan ! ack, i;
    cache[i] = WRI;
... fi }
... od
}
```

Developers of cache coherence protocols describe and reason about their protocols in terms of message passing, and, as these examples show, their reasoning can be directly expressed in Promela. Moreover, the proposed organization of Promela processes allows verification engineers to perform quick changes that are needed to reflect the modifications of the cache coherence protocol under verification that occur in the course of its development.

4. Parameterized Verification of Cache Coherence Protocols

The method for parameterized verification of cache coherence protocols presented in works [5, 6, 8] consists of two stages:

1. Performing the syntactical transformations of Promela models.
2. Refining the obtained model in accordance with the proposed procedure.

Model transformations have the following effect:

1. Reduction of the number of processes from n+1 (n cache controller processes and one system commutator process) to 4: two fully functioning cache controller processes, one abstract cache controller process that models the environment of the two processes, and the system commutator process. This transformation is possible due to the symmetry inherent in models of cache coherent protocols (all cache controller processes are identical and interchangeable, they do not have behaviors that depend on a particular process index value) and because the specification of cache coherence protocols only contains properties that regard the state of cache line in two caches.
2. Syntactical transformations of Promela operators constituting the model.

These transformations preserve invariant properties. This means that if such a property is true for the reduced model, then it is true for the initial model. A mathematical proof of the corresponding theorem is presented in articles [5, 6, 8].

4.1 Performing the Syntactical Transformations

The syntactical transformations presented in [5, 6, 8] may be performed manually. However, manual model modification is a very tedious, laborious and error-prone process. Moreover, some of the errors made may go undetected, as they will only lead to incorrect state space reduction and not to counterexamples. Therefore, it is highly desirable to perform the transformations automatically. To achieve that, I have developed a dedicated tool. With this tool, the verification engineer simply provides their Promela model as input to the tool, and the tool generates the transformed Promela model.

To automate the syntactical transformations, I have used a widespread approach to this kind of problems, according to which a tool builds the abstract syntax tree that represents the syntactical structure of the source code and then performs the transformations upon the tree traversal (Fig. 3).

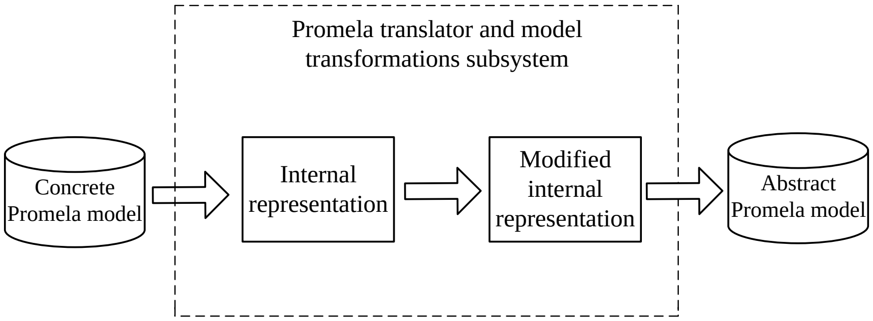

Promela translator and model transformations subsystem
Concrete Promela model
Internal representation
Modified internal representation
Abstract Promela model

Fig. 3. Scheme of Automated Model Transformation.

Abstract syntax trees are usually constructed by parsers. There are two ways of parser implementation: manual and by means of a parser generator tool (for example, Bison, ANTLR, Boost.Spirit). Due to the unnecessary complexity of the first approach, I have chosen the second one.

The Boost.Spirit library was chosen as the parser generator, because:

- Boost.Spirit promotes modern usage of the C++ language that allows us to work with abstractions, which are suitable for a given domain, without performance loss.
- Boost.Spirit eliminates the need for additional tools like Bison or ANTLR: The only tools needed are a C++ compiler and the Boost library.
- The grammars that Boost.Spirit accepts are attributed, which results in a very convenient way of abstract syntax tree generation.

- Boost.Spirit contains a number of built-in parsers.
- The generated parsers are very efficient [15].

The mechanism of synthesized and inherited attributes allows us to simplify the task of abstract syntax tree generation by dividing it into two sequentially performed subtasks:

1. Development of the grammar, testing and debugging of the grammar. During this step, we only need to focus on the question of whether the grammar can correctly determine the syntactical correctness of a Promela model.
2. Development of data structures for the nodes of the abstract syntax tree and definition of the types of attributes of the grammar rules. The attribute mechanism allows Boost.Spirit to generate abstract syntax trees automatically, without any need for the addition of node construction operators to the grammar.

Usage of the abstract syntax tree generated by Boost.Spirit as an intermediate representation of Promela models allowed us to divide the task of performing the syntactical transformations automatically into three subtasks:

1. Development of Promela grammar in the C++ language by means of Boost.Spirit.
2. Development of data structures for abstract syntax tree representation.
3. Development of algorithms for abstract syntax tree traversal and abstract model generation.

Promela grammar is presented in [16]. Its implementation in C++ using Boost.Spirit looks similarly to that description. However, as Boost.Spirit generates recursive descent parsers, I have eliminated left recursion from the grammar.

Data structures for the nodes of abstract syntax tree are developed according to the information that we want the nodes to represent and attribute propagation rules defined in Boost.Spirit's documentation. In the developed tool, data structures that correspond to the synthesized attributes of the Promela grammar rules, contain information about nonterminals that are part of the rules. This is a very straightforward and convenient way of implementation of these data structures. For example, the following rule that describes the nonterminal "module" of the Promela grammar

```
   qi::rule<Iter, module(), Skipper> module;
   module =
       proctype
       | init
       | ltl
       | utype
       | mtype
       | decl_lst
       | ';' ;
```

has a synthesized attribute of type `module`, which is implemented as follows:

```
   using module = boost::variant<
       proctype,
       init,
       ltl,
       utype,
       mtype,
       decl_lst
       >;
```

All the other nonterminals mentioned in this example have synthesized attributes of types implemented in a similar way.

The abstract syntax tree, which is generated automatically by Boost.Spirit based on the grammar and the attribute mechanism, consists of nodes of different types. Traversal of such tree is performed uniformly by means of visitors, as advocated by the Boost.Spirit documentation.

The syntactical transformations are performed during the abstract syntax tree traversal. I classified the transformations, most of which turned out to be in one of the three categories (transformations of assignments, transformations of expressions, transformations of communication actions), and precisely described them. To automatically carry them out, I have developed a number of abstract syntax tree modification algorithms and implemented them as part of the visitation mechanism. Printing out the modified syntax tree gives us the abstract Promela model.

For example, when generating the code for the abstract process, the following piece of Promela code

```
   proctype cache_controller(byte i) {
   do
   ...
   :: (cache[i] == M_MAU || cache[i] == M_MAU_I)
   && (message.opcode == wb_ready) ->
       final_ack_chan ! data, i;
       cache[i] = I
```

is transformed into

```
   proctype cache_controller_abs(byte i) {
   do
   ...
   :: true ->
```

```
        final_ack_chan ! data, i;
```

This example demonstrates the transformations of expressions and the assignment operator.

4.2 Abstraction Refinement

Execution of each type of initial requests consists of a particular sequence of events presented in the cache coherence protocol documentation. Considerations about the ordering of the events inspired the following refinement procedure:

1. For each type of initial requests define (according to the documentation) a partially ordered set $(A, \prec)$ of events ($\prec$ is a strict partial order):

   $\forall a_1, a_2 \in A: a_1 \prec a_2$, if action a_1 occurs earlier than action a_2.

2. While there are false counterexamples:

   2.1. Find action a that lead to the appearance of the counterexample. Find set A that contains action a: $a \in A$. In set A find action b such that $b \prec a$.

   2.2. Introduce a logical variable aux_b with the initial value $false$. In the model, replace b with the atomic sequence $b; aux_b \coloneqq true$.

3. By means of the logical AND, add aux_b to the guard of the command that contains action a. Replace a with the atomic sequence $a; aux_b \coloneqq false$.

For example, for one type of initial requests defined for the Elbrus-4C microprocessor, the set $(A, \prec)$ is as follows. Here, cc_i denotes the ith cache controller.

$\{a_0 =$ processing of the previous request from process $cc_i, 1 \le i \le n$ is finished,

$a_1 =$ requester cc_i sends an initial request,

$a_2 = system\_commutator$ receives the initial request,

$a_3 = system\_commutator$ sends snoop-requests to all $cc_j, 1 \le j \le n, j \ne i$,

$a_4 = cc_j$ receives a snoop-request, $1 \le j \le n, j \ne i$,

$a_5 = cc_j$ sends an answer to the snoop-request to the requester,

$a_6 =$ the requester receives the coherent answer from cc_j,

$a_7 =$ the requester sends the operation completion message to $system\_commutator$,

$a_8 = system\_commutator$ receives the operation completion message$\}$.

The relation $\prec$ is defined as follows: $\forall i, j = 0, \dots, |A| - 1: i < j \Rightarrow a_i \prec a_j$. We identify the auxiliary variables with the elements of the set A.

Refinement of the abstract model of the Elbrus-4C cache coherence protocol required us to introduce two auxiliary variables, because there were two spurious counterexamples. Let us examine the introduction of the first variable.

The analysis of the first counterexample showed that the abstract process had sent the operation completion message to $system\_commutator$ before $system\_commutator$ received a coherent answer. Examination of the set A allows us to conclude that action a_7 happening at the wrong time led to the counterexample. According to the refinement procedure, in the set A we find action a_6 and introduce an auxiliary variable `ack_received` with the initial value $false$. Then we replace the operator that corresponds to a_6 with the atomic sequence consisting of this operator and the operator that assigns $true$ to `ack_received`. After this, we add `ack_received` to the guard of the command of the abstract process that contains a_7 and replace the operator that corresponds to a_7 with the atomic sequence consisting of this operator and the operator that assigns $false$ to `ack_received`. Thus, we guarantee that the behavior of the abstract process that led the false counterexample will no longer be exhibited.

4.3 Verification Technique

According to the results obtained by the author in this and the previous works, the proposed verification technique consists of the following steps (Fig. 4):

1. Development of a concrete Promela model of the cache coherence protocol under verification. Using the proposed approach to model description, verification engineer develops Promela processes that model cache controllers and the system commutator and the necessary infrastructure elements (channel definitions, process creation). Specific actions performed by the processes correspond to the cache coherence protocol documentation.
2. Development of the abstract Promela model of the cache coherence protocol under verification. This step is performed automatically by the developed tool.
3. Verification of the abstract model. This step is the usual verification process of Promela models using the Spin model checker [17].
4. Analysis of the verification report generated by Spin. If there are no errors, then the verification process is finished with the conclusion that the cache coherence protocol is correct. If the report states the presence of an error, then the verification engineer should analyze the corresponding counterexample. If the engineer concludes that the counterexample is spurious because the corresponding sequence of steps is impossible in a real system, then the engineer refines the model in accordance with the proposed procedure and goes to step 3. Otherwise, if the counterexample represents an actual error in the cache coherence protocol, then the error is reported. When the protocol developers fix the error, the verification engineer incorporates the changes into the model and starts the verification process again (goes to step 1).

This sequence of steps is repeated until there are no counterexamples.

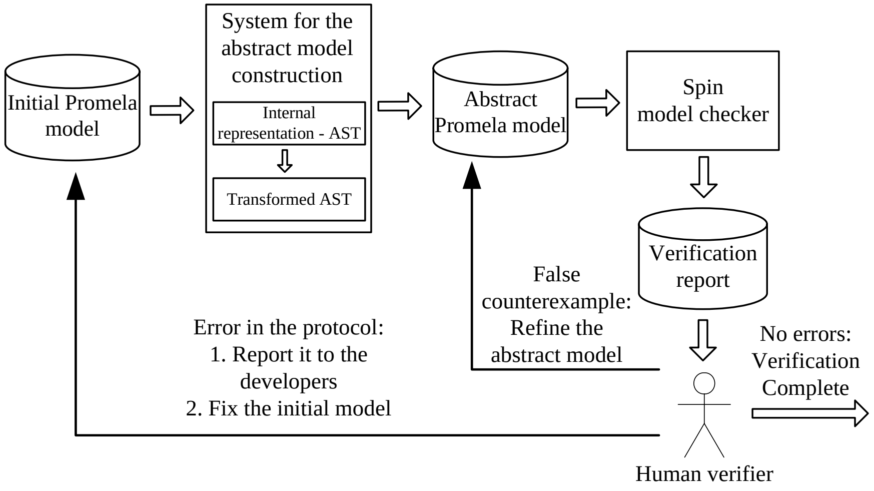

Initial Promela model
System for the abstract model construction
Internal representation - AST
Transformed AST
Abstract Promela model
Spin model checker
Verification report
False counterexample: Refine the abstract model
Error in the protocol: 1. Report it to the developers 2. Fix the initial model
No errors: Verification Complete
Human verifier

Fig. 4 Scheme of the Verification Process.

5. Experimental Results

The proposed method was used to verify the MOSI family cache coherence protocol implemented in the Elbrus-4C computer system. The abstraction refinement step was completed after the introduction of two auxiliary variables.

Table 1 and Table 2 show resources consumed for checking the property

$$\mathbf{G}\{\neg(cache[1] = M \wedge cache[2] = M)\},$$

respectively, on the original and the refined abstract model. Spin's optimization COLLAPSE was used. The experiments were performed on an Intel Xeon E5-2697 machine with a clock rate of 2.6 GHz and 264 Gb of RAM.

Table 1. Required resources — initial model

Number of cores	State space size	Memory consumption	Verification time
3	5.1×10^6	328 Mb	15 s
4	1.3×10^9	81 Gb	1.5 h

Table 2. Required resources — abstract model

Number of cores	State space size	Memory consumption	Verification time
any > 2	2.2×10^6	108 Mb	6.2 s

Tables 1 and 2 show that even for $n = 3$ there is a gain in state space size and memory consumption. The needed amount of manual work is acceptable. Meanwhile, verification of the constructed abstract model means verification of the protocol for any $n \geq 3$. The task has been reduced to checking of $\sim 10^6$ states, which consumes ~ 100 Mb of memory.

6. Conclusion

Many high-performance computers and most multicore microprocessors use shared memory and utilize complicated caching mechanisms. To ensure that multiple copies of data are kept up-to-date, cache coherence protocols are employed. Errors in the protocols and their implementations may cause serious consequences such as data corruption and system hanging. This explains the urgency of the corresponding verification methods.

The main problem when verifying cache coherence protocols (and other systems with a large number of components) by a fully automated method of model checking is state explosion. The article proposes a technique to overcome the problem for cache coherence protocols and make verification scalable. The price paid for scalability is acceptable, because the main ingredient – the verification method – is highly automated by the developed tool. Part of the method that requires manual work, namely, model refinement, can be done with a reasonable amount of effort, as shown by means of the Elbrus-4C protocol verification example. An approach to describing protocol models in Promela, a widely spread language in the verification community, is proposed. This approach lets us reflect the way protocol designers talk about protocols by representing protocols as a set of communicating finite-state machines.

The technique was successfully applied to the verification of the MOSI family cache coherence protocols implemented in the Elbrus-4C computer system.

Directions for future research include:

1. Development of methods and tools for verification of cache coherence protocols that are implemented by multiple levels of cache. The newest microprocessors (for example, Elbrus-8C, which employs the second- and third-level caches to implement cache coherence) define the need for such methods and tools.

2. Development of methods and tools for verification of hardware implementations of cache coherence protocols. In this direction, I have developed a tool that generates assembly code based on Promela models of cache coherence protocols. With this tool, I have found several dozen errors in the implementation of cache coherence in Elbrus microprocessors. Still, further research is needed to increase the level of confidence in design correctness.

References

[1]. Patterson D.A., Hennessy J.L. Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann, 2013. 800 p.

[2]. Sorin D.J., Hill M.D., Wood D.A. A Primer on Memory Consistency and Cache Coherence. Morgan and Claypool, 2011. 195 p.

[3]. Clarke E.M., Grumberg O., Peled D.A. Model Checking. MIT Press, 1999. 314 p.
[4]. Burenkov V.S. Analiz primenimosti instrumenta Spin k verifikacii protokolov kogerentnosti pamyati [An analysis of the Spin model checker applicability to cache coherence protocols verification]. Voprosy radioehlektroniki. Ser. EVT. Issues of Radioelectronics, the series EVT], 2014, issue. 3, pp. 126-134 (in Russian).
[5]. Burenkov V.S., Kamkin A.S. Checking Parameterized PROMELA Models of Cache Coherence Protocols. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 4, 2016, pp. 57-76. DOI: 10.15514/ISPRAS-2016-28(4)-4
[6]. Burenkov V.S., Kamkin A.S. Metod masshtabiruemoi verifikacii PROMELA-modelei protocolov kogerentnosti kesh-pamyati [A Method for Scalable Verification of PROMELA Models of Cache Coherence Protocols]. Sb. trudov VII Vserossiiskoi nauchno-technicheskoi konferencii “Problemi razrabotki perspectivnih micro- i nanoelektronnih sistem” [Proceedings of the VII All-Russian Scientific and Technical Conference "Problems of Development of Advanced Micro- and Nanoelectronic Systems"]. 2016, part II, pp. 54-60 (in Russian).
[7]. Burenkov V.S., Kamkin A.S. Applying Parameterized Model Checking to Real-Life Cache Coherence Protocols. Proc. of IEEE East-West Design & Test Symposium. 2016. pp. 1-4.
[8]. Burenkov V.S., Ivanov S.R. Metod postroeniya abstraktnih modelei, ispolzuemih dlya verifikacii protocolov kogerentnosti kesh-pamyati [A Method for Construction of Abstract Models Used for Verification of Cache Coherence Protocols]. Vestnik MGTU im N.E. Baumana [Herald of the Bauman Moscow State Technical University], 2017, issue 1, pp. 49-66 (in Russian).
[9]. McMillan K. Parameterized Verification of the FLASH Cache Coherence Protocol by Compositional Model Checking. Conference on Correct Hardware Design and Verification Methods, 2001. pp. 179-195.
[10]. Chou C.-T., Mannava P.K., Park S. A Simple Method for Parameterized Verification of Cache Coherence Protocols. Formal Methods in Computer-Aided Design, 2004. LNCS, Vol. 3312, pp. 382-398.
[11]. Krstic S. Parameterized System Verification with Guard Strengthening and Parameter Abstraction. International Workshop on Automated Verification of Infinite-State Systems, 2005.
[12]. Talupur M., Tuttle M.R. Going with the Flow: Parameterized Verification Using Message Flows. Formal Methods in Computer-Aided Design, 2008. pp. 1-8.
[13]. O'Leary J., Talupur M., Tuttle M.R. Protocol Verification Using Flows: An Industrial Experience. Formal Methods in Computer-Aided Design, 2009. pp. 172-179.
[14]. Baier C., Katoen J.-P. Principles of Model Checking. The MIT Press. 2008. 984 p.
[15]. de Guzman, J. Fastest numeric parsers in the world! http://boost-spirit.com/home/2014/09/03/fastest-numeric-parsers-in-the-world/.
[16]. Spin Version 6 – Promela Grammar. http://spinroot.com/spin/Man/grammar.html.
[17]. Holzmann, G. The Spin Model Checker: Primer and Reference Manual. Addison-Wesley. 2004. 608 p.

Методика параметризованной верификации протоколов когерентности памяти

В.С. Буренков <burenkov_v@mcst.ru>,
АО «МЦСТ», 119334, Россия, г. Москва, ул. Вавилова, 24

Аннотация. В статье представлена методика масштабируемой функциональной верификации протоколов когерентности памяти, которая основана на методе верификации, который ранее был разработан автором статьи. Масштабируемость при верификации означает независимость работ по верификации от размера модели, то есть от количества процессоров верифицируемой системы. В статье предложен подход к разработке формальных моделей протоколов когерентности памяти на языке Promela. Приведены примеры описаний, взятые из модели протокола когерентности памяти системы Эльбрус-4С. Результирующие формальные модели отражают представление протоколов когерентности памяти, используемое разработчиками таких протоколов – в виде множества взаимодействующих конечных автоматов. Описана разработка программного инструмента, написанного на языке C++ с использованием библиотеки Boost.Spirit в качестве генератора синтаксических анализаторов. Программный инструмент позволяет автоматизировать выполнение синтаксических преобразований Promela-моделей. Выполнение данных синтаксических преобразований происходит в соответствии с методом верификации. В статье представлена процедура уточнения модифицированных моделей, основанная на введении в модель вспомогательных переменных. Использовать эту процедуру предлагается в том случае, когда при верификации возникают ложные сообщения об ошибках, для устранения таких сообщений. Представлена методика верификации, которая состоит из следующих шагов: разработка исходной модели протокола когерентности памяти на языке Promela, автоматизированное преобразование данной модели согласно методу верификации, верификация модифицированной модели с помощью инструментального средства Spin, анализ отчета о верификации, сгенерированного инструментом Spin. Изложенная методика была успешно применена для верификации протокола когерентности памяти семейства MOSI, реализованного в микропроцессорной системе Эльбрус-4С. Экспериментальные результаты показали, что затраты процессорного времени и памяти на проведение параметризованной верификации незначительны, а требуемый объем ручной работы приемлем. Для уточнения модифицированной модели протокола системы Эльбрус-4С понадобилось ввести лишь две вспомогательные переменные.

Ключевые слова: многоядерные микропроцессоры; мультипроцессоры с разделяемой памятью; протоколы когерентности памяти; проверка моделей; Spin; Promela.

DOI: 10.15514/ISPRAS-2017-29(4)-15

Для цитирования: Буренков В.С. Методика параметризованной верификации протоколов когерентности памяти. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 231-246 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-15

Список литературы

[1]. Patterson D.A., Hennessy J.L. Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann, 2013. 800 p.

[2]. Sorin D.J., Hill M.D., Wood D.A. A Primer on Memory Consistency and Cache Coherence. Morgan and Claypool, 2011. 195 p.

[3]. Clarke E.M., Grumberg O., Peled D.A. Model Checking. MIT Press, 1999. 314 p.

[4]. Буренков В.С. Анализ применимости инструмента Spin к верификации протоколов когерентности памяти. Вопросы радиоэлектроники. Сер. ЭВТ. 2014. Вып. 3. стр. 126-134.

[5]. Burenkov V.S., Kamkin A.S. Checking Parameterized PROMELA Models of Cache Coherence Protocols. Trudy ISP RAN/Proc. ISP RAS. vol. 28, issue 4, 2016, pp. 57-76. DOI: 10.15514/ISPRAS-2016-28(4)-4

[6]. Буренков В.С., Камкин А.С. Метод масштабируемой верификации PROMELA-моделей протоколов когерентности кэш-памяти. Сб. трудов VII Всероссийской научно-технической конференции «Проблемы разработки перспективных микро- и наноэлектронных систем». 2016. Часть II. стр. 54-60.

[7]. Burenkov V.S., Kamkin A.S. Applying Parameterized Model Checking to Real-Life Cache Coherence Protocols. Proc. of IEEE East-West Design & Test Symposium. 2016. pp. 1-4.

[8]. Буренков В.С., Иванов С.Р. Метод построения абстрактных моделей, используемых для верификации протоколов когерентности кэш-памяти. Вестник МГТУ им. Н.Э. Баумана. 2017, вып. 1, стр. 49-66.

[9]. McMillan K. Parameterized Verification of the FLASH Cache Coherence Protocol by Compositional Model Checking. Conference on Correct Hardware Design and Verification Methods, 2001. pp. 179-195.

[10]. Chou C.-T., Mannava P.K., Park S. A Simple Method for Parameterized Verification of Cache Coherence Protocols. Formal Methods in Computer-Aided Design, 2004. LNCS, Vol. 3312, pp. 382-398.

[11]. Krstic S. Parameterized System Verification with Guard Strengthening and Parameter Abstraction. International Workshop on Automated Verification of Infinite-State Systems, 2005.

[12]. Talupur M., Tuttle M.R. Going with the Flow: Parameterized Verification Using Message Flows. Formal Methods in Computer-Aided Design, 2008. pp. 1-8.

[13]. O'Leary J., Talupur M., Tuttle M.R. Protocol Verification Using Flows: An Industrial Experience. Formal Methods in Computer-Aided Design, 2009. pp. 172-179.

[14]. Baier C., Katoen J.-P. Principles of Model Checking. The MIT Press. 2008. 984 p.

[15]. de Guzman, J. Fastest numeric parsers in the world! http://boost-spirit.com/home/2014/09/03/fastest-numeric-parsers-in-the-world/.

[16]. Spin Version 6 – Promela Grammar. http://spinroot.com/spin/Man/grammar.html.

[17]. Holzmann, G. The Spin Model Checker: Primer and Reference Manual. Addison-Wesley. 2004. 608 p.

Test Generation for Digital Hardware Based on High-Level Models

[1] *M.M. Chupilko <chupilko@ispras.ru>*
[1,2.3] *A.S. Kamkin <kamkin@ispras.ru>*
[1] *M.S. Lebedev <lebedev@ispras.ru>*
[1] *S.A. Smolov <smolov@ispras.ru>*
[1] *Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*
[2] *Lomonosov Moscow State University (MSU), GSP-1, Leninskie Gory, Moscow, 119991, Russia.*
[3] *Moscow Institute of Physics and Technology (MIPT), 9, Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russia*

Abstract. Hardware testing is a process aimed at detecting manufacturing faults in integrated circuits. To measure test quality, two main metrics are in use: fault detection abilities (fault coverage) and test application time (test length). Many algorithms have been suggested for test generation; however, no scalable solution exists. In this paper, we analyze applicability of functional tests generated from high-level models for low-level manufacturing testing. A particular test generation method is considered. The input information is an HDL description. The key steps of the method are system model construction and coverage model construction. Both models are automatically extracted from the given description. The system model is a representation of the design in the form of high-level decision diagrams. The coverage model is a set of LTL formulae defining reachability conditions for the transitions of the extended finite state machine. The models are translated into the input format of a model checker. For each coverage model formula the model checker generates a counterexample, i.e. an execution that violates the formula (makes the corresponding transition to fire). The approach is intended for covering of all possible execution paths of the input HDL description and detecting dead code. Experimental comparison with the existing analogues has shown that it produces shorter tests, but they achieve lower stuck-at fault coverage comparing with the dedicated approach. An improvement has been proposed to overcome the issue.

Keywords: digital hardware; hardware description language; manufacturing testing; stuck-at fault; high-level decision diagram; extended finite state machine; model checking; fault propagation.

DOI: 10.15514/ISPRAS-2017-29(4)-16

For citation: Chupilko M.M., Kamkin A.S., Lebedev M.S., Smolov S.A. Test Generation for Digital Hardware Based on High-Level Models. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 247-256. DOI: 10.15514/ISPRAS-2017-29(4)-16

1. Introduction

Functional verification and test generation are resource-consuming activities of the hardware design process [1]. To automate these activities, *models* are frequently used. Models are mathematical abstractions that describe system structure and behavior. There is a variety of verification and test generation problems that can be solved with the help of models: checking system behavior in simulation-based verification [2], directed test generation [3], etc.

The essential stage of the hardware design process is *register-transfer-level* (*RTL*) design. This stage results in code in a *hardware description language* (*HDL*), such as VHDL and Verilog [4]. The RTL model is automatically synthesized into a *gate-level netlist* represented in an HDL or a special language, such as BLIF [5]. Finally, the place-and-route stage is applied to produce a chip layout.

Functional verification, including functional test generation, deals with RTL models, while generation of manufacturing tests uses gate-level netlists. In this paper, we analyze applicability of functional tests for manufacturing testing. The motivation is clear: the simpler the model, the easier to get tests. We extract high-level models from HDL descriptions and generate tests from them. The approach allows reaching good code coverage with short tests [6].

This paper continues research initiated in [7], where we compared fault detection abilities of different test generation methods. A test is said to *detect* a fault, if the *mutant*, i.e. the design with the injected fault, and the original design return different outputs for the test's input sequence. Fault detection ability is measured as the amount of faults having been detected.

The rest of the paper is organized as follows. Section 2 defines formalisms used in the work and gives a brief overview of a fault model. Section 3 summarizes works on applying model-based techniques to manufacturing testing. Section 4 describes the proposed approach. Section 5 reports experimental results. Section 6 suggests a possible approach improvement. Section 7 discusses the results of the work and concludes the paper.

2. Preliminaries

Let V be a finite set of *variables*. A *valuation* is a function that associates each variable with a value from the corresponding domain. Let D_V be the set of all possible valuations of V.

A *guard* is a Boolean function defined on valuations: $D_V \to \{0, 1\}$. An *action* is a transformation of valuations: $D_V \to D_V$. A pair $\gamma \to \delta$, where γ is a guard and δ is an action, is called a *guarded action*. It is implied that there is a description of every function in an HDL-like language (thus, we can reason not only about semantics, but about syntax as well).

An *extended finite state machine* (EFSM) is a tuple $M = \langle S_M, V_M, T_M \rangle$, where S_M is a set of *states*, $V_M = (I_M \cup O_M \cup R_M)$ is a set of variables, consisting of *inputs* (I_M), *outputs* (O_M) and *registers* (R_M), and T_M is a set of *transitions*. A transition $t \in T_M$ is a tuple $(s_t, \gamma_t \to \delta_t, s'_t)$, where s_t and s'_t are respectively the *initial* and the *final* state of t, whereas γ_t and δ_t are respectively the guard and the action of t.

A pair $(s, \nu) \in S_M \times D_{V_M}$ is referred to as a *configuration*. A transition t is said to be *enabled* in a configuration (s, ν) if $s_t = s$ and $\gamma_t(\nu) = 1$.

An EFSM operates in discrete time. In the beginning, it resets the configuration: $(s, \nu) \coloneqq (s_0, \nu_0)$, where (s_0, ν_0) is a predefined configuration. On every tick, it computes the set of enabled transitions:

$$T_E \coloneqq \{t \in T_M | (s_t = s) \wedge (\gamma_t(\nu) = 1)\}.$$

A single transition $t \in T_E$ (chosen nondeterministically) fires: $(s, \nu) \coloneqq \left(s'_t, \delta_t(\nu)\right)$.

A *netlist* is a tuple $N = \langle V_N, G_N, L_N \rangle$, where V_N is a set of variables, G_N is a set of *gates*, and L_N is a set of *latches*. A gate $g \in G_N$ is a tuple $\left(I_g, o_g, f_g\right)$, where $I_g \subseteq V_N$ and $o_g \in V_N$ are respectively the *inputs* and the *output* of g, and $f_g\colon Dom_{I_g} \to \{0, 1\}$ is the function of g. A latch $l \in L_N$ is a tuple (i_l, o_l), where $i_l \in V_N$ and $o_l \in V_N$ are respectively the *input* and the *output* of l.

A netlist operates as follows. In the beginning, it initializes the latches' outputs with some predefined values. On every tick, it computes the gates' output values based on the input values and transmits the latches' input values to the outputs.

To compare test generation methods, the well-known *stuck-at fault* model is used. We consider the following variation of the model. There is a stuck-at fault if some gate is "corrupted" so as its function, which is not identically equal to a constant, always returns a constant, either 0 (stuck-at-0) or 1 (stuck-at-1).

3. Related work

This section overviews the existing model-based test generation methods aimed at covering stuck-at faults.

In [8], an approach to functional test generation for VHDL designs is proposed. The method consists of the following stages:

1) translation of an HDL description into *binary decision diagrams* (*BDD*);
2) insertion of a stuck-at fault into the BDD;
3) generation of a *distinguishing test* for the original BDD and the faulty one.

For the HDL-to-BDD translation, the approach uses a method described in [9].

In [10], a combined approach is suggested. It uses two kinds of models: a *high-level decision diagram* (*HLDD*) and an EFSM. Both models are automatically extracted from an HDL description. HLDD is a generalization of BDD: non-terminal nodes of a diagram are marked not only by 0 and 1 but by arbitrary expressions. First, a test is generated that covers all branches of the diagram. Then, the test is passed to the EFSM simulator to measure the transition coverage. To cover the uncovered EFSM transitions, a special *backjumping* technique is applied.

In [6], another EFSM-based approach is proposed. It fixes several issues of the previously mentioned method and uses a different EFSM extraction technique. The experiments have shown that new tests are shorter, while code coverage is the same.

In [7], the method [6] is experimentally compared with another one, which uses the ABC equivalence checker [11] to generate a distinguishing sequence for two BLIF descriptions. The EFSM-based method demonstrates higher HDL code coverage and shorter tests, while the ABC-based one achieves higher stuck-at fault coverage.

4. Proposed approach

In this paper, we continue our work on applying the model-checking techniques for test generation [12]. The approach allows achieving high HDL code coverage with very short tests. Our current goal is to evaluate how good the approach is in terms of the stuck-at-fault coverage. The method flow is shown in Fig. 1.

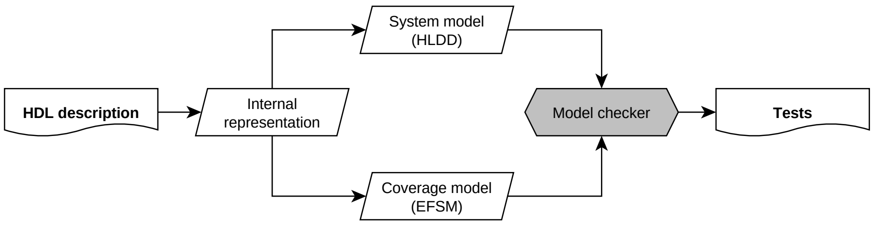


Fig. 1. Model checking-based approach to test generation for HDL descriptions

The method uses two models extracted from an HDL description: a *system model*, which is based on the HLDD formalism, and a *coverage model*, which utilizes the EFSM concept (see [12] and [13] for more details). The system model represent the system functionality, while the coverage model defines a set of conditions, so-called *coverage items*, to be covered by tests.

Let us say a few words about the coverage model. For each HDL process, a separate EFSM is extracted. The EFSM states are mutually disjoint constraints on *state-like registers* (*SLR*). The SLR are chosen automatically with the help of dataflow-based heuristics. The EFSM transitions are constructed from the process execution paths. Coverage items are *reachability conditions* for the EFSM transitions. Let s be an EFSM's state, $c(s)$ be the corresponding constraint, and t be an outgoing transition, i.e. $s_t = s$. In terms of the *linear temporal logic* (*LTL*), the reachability condition is as follows: $\mathbf{F}\{c(s) \wedge \gamma_t\}$, where $\mathbf{F}\{x\}$ means that x will eventually be true.

In accordance with [12], the system model and the coverage items in the negated form ($\neg\mathbf{F}\{c(s) \wedge \gamma_t\}$), are translated into the input format of a model checker. For each item, the model checker constructs a *counterexample*, i.e. an execution that violates the corresponding formula. Since coverage is formulated in the negated form, the counterexamples are executions that reach the related EFSM transitions. Counterexamples are translated into testbenches and executed by an HDL simulator.

The method is aimed at covering EFSM transitions. However, being rather flexible, it can be applied to various coverage models.

5. Experiments

The proposed approach has been implemented in the Retrascope 0.2.1 tool [14]. The implementation uses the Fortress library [15] and the nuXmv model checker [16].

The method has been tested on some designs from the ITC'99 benchmark [17]. Three test generation methods were compared:

1) the method described in this paper (nuXmv);
2) the method based on EFSM traversal (RETGA) [6];
3) the method based on equivalence checking (ABC) [7].

The third method uses the ABC tool [11] to generate distinguishing sequences for design represented in the BLIF format.

Two metrics were used for test comparison: the length in ticks and the number of killed mutants (detected faults).

To generate mutants, a DTESK prototype was used. Given a fault model and an HDL description, the tool generates a number of mutants along with testbenches. Each testbench contains the original design and the mutant; it reads input values from the file, passes them to both designs, and compares the outputs' values; if there is a mismatch at some tick, then the mutant is considered to be *killed*.

Table 1 shows information about the ITC'99 designs: the source code size (in lines of code), the system model size, and the number of stuck-at fault mutants.

Table 1. ITC'99 designs

Design	***Source code***	***System model***	***Number of mutants***
B01	102	207	88
B02	70	143	48
B04	101	809	1342
B06	127	442	94
B07	92	370	784
B08	88	315	324
B09	100	263	284

Table 2 shows the test-related information: the length in ticks and the percentage of the killed mutants.

Table 2. Test generation methods comparison

Design	***ABC (ticks)***	***RETGA (ticks)***	***nuXmv (ticks)***	***ABC (%)***	***RETGA (%)***	***nuXmv (%)***
B01	227	49	37	90.91	98.86	90.9
B02	86	33	28	0	0	0
B04	—	36	56	—	99.93	99.93
B06	100	76	63	100	100	100
B07	133	166	162	0	0	0
B08	2745	52	31	98.77	79.94	44.44
B09	777	231	55	97.18	0	0

Comparison results are as follows. For some designs (B02 and B07), all methods achieve 0% stuck-at fault coverage. Such designs are classified as *untestable* [18]; their outputs are calculated in such a way that the internal stuck-at faults cannot affect their outputs. For some designs (B01, B04, and B06), the proposed method reaches the same or comparable stuck-at fault coverage as the leading one. Note that in such cases model-checking-based tests are usually shorter than tests produced by other methods. Finally, there are some designs (B08 and B09), where both nuXmv and RETGA reach lower stuck-at fault coverage than ABC. Additional efforts are needed to cope with this issue. A possible improvement is described below.

6. Future improvements

In our opinion, the main drawback of the proposed method and similar approaches is a lack of fault propagation. Broadly speaking, an EFSM model contains a stuck-at fault if some assignment ($v \coloneqq RHS$) of some transition is "corrupted" (RHS is substituted by 0 or 1). To activate the fault, a test should cause the transition to fire; however, this is not enough. The erroneous values should be propagated to the model outputs. Thus, the coverage model should be extended.

Given an EFSM model M, let us make some definitions. A variable v is *defined* in a transition x ($v \in Def_x$) if δ_x contains an assignment to v. A variable v is *used* in a transition y ($v \in Use_y$) if v appears either in γ_y or in the right-hand side of δ_y. A transition y *depends* on a transition x ($y \in DEP(x)$) if $Def_x \cap Use_y \neq \emptyset$. Depending on how v is used, in γ_y or in δ_y, they say about a *control dependency* ($y \in DEP_c(x)$) or a *data dependency* ($y \in DEP_d(x)$) respectively.

A *propagation path* for a transition t is a sequence of transitions $\{t_i\}_{i=0}^{n}$ such that:

1) $t_0 = t$;

2) $t_{i+1} \in DEP_d(t_i)$, for all $0 \leq i < n$;

3) $Def_{t_n} \cap O_M \neq \emptyset$.

Given a propagation path $\{(s_i, \gamma_i \to \delta_i, s_i')\}_{i=0}^{n}$, the propagation condition can be expressed as follows:

$$\varphi = \mathbf{F}\Big\{(c(s_0) \wedge \gamma_0) \wedge \mathbf{F}\big\{(c(s_1) \wedge \gamma_1) \wedge \mathbf{F}\{\dots \mathbf{F}\{c(s_n) \wedge \gamma_n\} \dots\}\big\}\Big\}.$$

Note that the notion of propagation path and the propagation condition can be refined so as to avoid variable redefinitions and other undesirable effects.

If there are no propagation paths for a given transition, the original coverage item, $\varphi_0 \equiv \mathbf{F}\{c(s_0) \wedge \gamma_0\}$, is removed. If there are multiple propagation paths, two main strategies can be applied:

1) *try all of the propagation paths*:
    a. split the coverage item φ_0 into the set of all possible fault propagation conditions: $\{\varphi_1, \dots, \varphi_m\}$;
2) *try at least one of the propagation paths*:
    a. replace the coverage item φ_0 with the disjunction $\bigvee_{i=1}^{m} \varphi_m$.

7. Conclusion

The primary scope of this work is reusing functional tests for manufacturing testing. The paper describes a high-level test generation approach and analyzes whether it is effective in detecting low-level faults. The approach implements two important facilities: automatic extraction of models from HDL descriptions and directed test generation based on model checking. The method is extremely flexible and can be customized by choosing a proper coverage model. The presented implementation tends to produce short tests with mediocre stuck-at fault coverage. We think that fault detection abilities of the approach can be increased by adding fault propagation conditions into coverage items. This may serve as a topic for future research.

Acknowledgment

The authors would like to thank Russian Foundation for Basic Research (RFBR). The reported study was supported by RFBR research project № 15-07-03834.

References

[1]. Bergeron J. Writing Testbenches: Functional Verification of HDL Models. Springer, 2003, 478 p. DOI: 10.1007/978-1-4615-0302-6.

[2]. Ivannikov V.P., Kamkin A.S., Kossatchev A.S., Kuliamin V.V., Petrenko A.K. The Use of Contract Specifications for Representing Requirements and for Functional Testing of Hardware Models. Programming and Computer Software, 33(5), 2007, pp. 272-282. DOI: 10.1134/s0361768807050039.

[3]. Mishra P., Dutt N. Specification-Driven Directed Test Generation for Validation of Pipelined Processors. ACM Transactions on Design. Automation of Electronic Systems, 13(3), 2008, pp 1-36. DOI: 10.1145/1367045.1367051.

[4]. Botros N.M. HDL Programming Fundamentals: VHDL and Verilog. Charles River Media, 2005, 506 p.

[5]. Berkeley Logic Interchange Format (BLIF). Berkeley, U.C., Oct Tools Distribution 2, 1992, pp. 197-247.
[6]. Melnichenko I., Kamkin A., Smolov S. An Extended Finite State Machine-Based Approach to Code Coverage-Directed Test Generation for Hardware Designs. Proceedings of ISP RAS, 2015, 27(3), pp. 161-182. DOI: 10.15514/ispras-2015-27(3)-12.
[7]. Smolov S., Lopez J., Kushik N., Yevtushenko N., Chupilko M., Kamkin A. Testing Logic Circuits at Different Abstraction Levels: An Experimental Evaluation. Proceedings of IEEE East-West Design Test Symposium (EWDTS), 2016, pp. 189-192. DOI: 10.1109/ewdts.2016.7807687.
[8]. Ferrandi F., Fummi F., Gerli L., Sciuto D. Symbolic Functional Vector Generation for VHDL Specifications. Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 1999, pp. 442-446. DOI: 10.1145/307418.307541.
[9]. Minato S. Generation of BDDs from Hardware Algorithm Descriptions. Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 1996, pp. 644-649. DOI: 10.1109/iccad.1996.571340.
[10]. Guglielmo G.D., Fummi F., Jenihhin M., Pravadelli G., Raik J., Ubar R. On the Combined Use of HLDDs and EFSMs for Functional ATPG. Proceedings of IEEE East-West Design and Test Symposium (EWDTS), 2007, pp. 503-508.
[11]. Brayton R., Mishchenko A. ABC: An Academic Industrial-Strength Verification Tool. Proceedings of the Computer Aided Verification Conference (CAV), 2010, pp. 24-40. DOI: 10.1007/978-3-642-14295-6_5.
[12]. Kamkin A., Lebedev M., Smolov S. An EFSM-Driven and Model Checking-Based Approach to Functional Test Generation for Hardware Designs. Proceedings of IEEE East-West Design and Test Symposium (EWDTS), 2016, pp. 60-63. DOI: 10.1109/ewdts.2016.7807736.
[13]. Smolov S., Kamkin A. A Method of Extended Finite State Machines Construction From HDL Descriptions Based on Static Analysis of Source Code. St. Petersburg State Polytechnical University Journal. Computer Science, Telecommunications, 1(212), 2015, pp. 60-73 (in Russian). DOI: 10.5862/jcstcs.212.6.
[14]. Retrascope site. http://forge.ispras.ru/projects/retrascope
[15]. Fortress library site. http://forge.ispras.ru/projects/solver-api
[16]. Cavada D., Cimatti A., Dorigatti M., Griggio A., Mariotti A., Micheli A., Mover S., Roveri M., Tonetta S. The nuXmv symbolic model checker. Proceedings of the 16th International Conference on Computer Aided Verification (CAV), No.8559, 2014, pp. 334-342. DOI: 10.1007/978-3-319-08867-9_22.
[17]. ITC'99 benchmark site. http://www.cad.polito.it/tools/itc99.html
[18]. Liu X., Hsiao M.S. On Identifying Functionally Untestable Transition Faults. Proceedings of the Ninth IEEE International High-Level Design Validation and Test Workshop, 2004, pp. 121-126. DOI: 10.1109/hldvt.2004.1431252.

Генерация тестов для цифровой аппаратуры на основе высокоуровневых моделей

[1] *М.М. Чупилко <chupilko@ispras.ru>*
[1,2.3] *А.С. Камкин <kamkin@ispras.ru>*
[1] *М.С. Лебедев <lebedev@ispras.ru>*
[1] *С.А. Смолов <smolov@ispras.ru>*
[1] *Институт системного программирования 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.
[3] *Московский физико-технический институт,*
141701, Россия, Московская обл., г. Долгопрудный, Институтский пер., д. 9.

Аннотация. Тестирование аппаратуры — это процесс, нацеленный на обнаружение неисправностей, внесенных в интегральные схемы в процессе производства. Для оценки качества таких тестов используют две основные метрики: способность обнаруживать ошибки (покрытие ошибок) и время тестирования (длина теста). Известно множество методов генерации тестов, однако масштабируемого решения, применимого к сложной цифровой аппаратуре, нет до сих пор. В данной статье анализируется возможность использования функциональных тестов, построенных по высокоуровневым моделям (прежде всего, моделям уровня регистровых передач), для низкоуровневого производственного тестирования. Рассматривается конкретный метод генерации тестов, использующий технику проверки моделей (model checking). Входной информацией выступает HDL-описание. Метод состоит из двух ключевых шагов: построение модели системы и построение модели покрытия. Указанные модели автоматически извлекаются из HDL-описания. Модель системы представлена в виде высокоуровневых решающих диаграмм. Модель покрытия — это множество LTL-формул, определяющих условия достижимости переходов расширенного конечного автомата, описывающего систему. Построенные модели транслируются во входной формат инструмента проверки моделей (model checker), который для каждой формулы модели покрытия генерирует контрпример — вычисление, нарушающее эту формулу, то есть приводящее к срабатыванию соответствующего перехода автомата. Изначально рассматриваемый метод предназначался для покрытия всех путей исполнения HDL-описания и обнаружения недостижимого кода. Экспериментальное сравнение метода с существующими аналогами показало, что он строит более короткие тесты, однако эти тесты достигают меньшего уровня покрытия константных неисправностей, чем тесты, построенные с помощью специальных средств. В статье предлагается модификация метода для преодоления указанного недостатка.

Ключевые слова: цифровая аппаратура; язык описания аппаратуры; производственное тестирование; константная ошибка; высокоуровневая решающая диаграмма; расширенный конечный автомат; проверка модели; дерево распространения.

DOI: 10.15514/ISPRAS-2017-29(4)-16

Для цитирования: Чупилко М.М., Камкин А.С., Лебедев М.С., Смолов С.А. Метод генерации тестов для цифровой аппаратуры, основанный на высокоуровневых моделях. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 247-256 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-16

Список литературы

[1]. Bergeron J. Writing Testbenches: Functional Verification of HDL Models. Springer, 2003, 478 p. DOI: 10.1007/978-1-4615-0302-6.

[2]. Иванников В.П., Камкин А.С., Косачев А.С., Кулямин В.В., Петренко А.К. Использование контрактных спецификаций для представления требований и функционального тестирования моделей аппаратуры. Программирование, т. 33, № 5, 2007 г., стр. 272-282.

[3]. Mishra P., Dutt N. Specification-Driven Directed Test Generation for Validation of Pipelined Processors. ACM Transactions on Design. Automation of Electronic Systems, 13(3), 2008, pp 1-36. DOI: 10.1145/1367045.1367051.

[4]. Botros N.M. HDL Programming Fundamentals: VHDL and Verilog. Charles River Media, 2005, 506 p.

[5]. Berkeley Logic Interchange Format (BLIF). Berkeley, U.C., Oct Tools Distribution 2, 1992, pp. 197-247.

[6]. Melnichenko I., Kamkin A., Smolov S. An Extended Finite State Machine-Based Approach to Code Coverage-Directed Test Generation for Hardware Designs. Proceedings of ISP RAS, 2015, 27(3), pp. 161-182. DOI: 10.15514/ispras-2015-27(3)-12.

[7]. Smolov S., Lopez J., Kushik N., Yevtushenko N., Chupilko M., Kamkin A. Testing Logic Circuits at Different Abstraction Levels: An Experimental Evaluation. Proceedings of IEEE East-West Design Test Symposium (EWDTS), 2016, pp. 189-192. DOI: 10.1109/ewdts.2016.7807687.

[8]. Ferrandi F., Fummi F., Gerli L., Sciuto D. Symbolic Functional Vector Generation for VHDL Specifications. Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 1999, pp. 442-446. DOI: 10.1145/307418.307541.

[9]. Minato S. Generation of BDDs from Hardware Algorithm Descriptions. Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 1996, pp. 644-649. DOI: 10.1109/iccad.1996.571340.

[10]. Guglielmo G.D., Fummi F., Jenihhin M., Pravadelli G., Raik J., Ubar R. On the Combined Use of HLDDs and EFSMs for Functional ATPG. Proceedings of IEEE East-West Design and Test Symposium (EWDTS), 2007, pp. 503-508.

[11]. Brayton R., Mishchenko A. ABC: An Academic Industrial-Strength Verification Tool. Proceedings of the Computer Aided Verification Conference (CAV), 2010, pp. 24-40. DOI: 10.1007/978-3-642-14295-6_5.

[12]. Kamkin A., Lebedev M., Smolov S. An EFSM-Driven and Model Checking-Based Approach to Functional Test Generation for Hardware Designs. Proceedings of IEEE East-West Design and Test Symposium (EWDTS), 2016, pp. 60-63. DOI: 10.1109/ewdts.2016.7807736.

[13]. Смолов С., Камкин А. Метод построения расширенных конечных автоматов по HDL-описанию на основе статического анализа кода. Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление, 1(212), 2015, стр. 60-73. DOI: 10.5862/jcstcs.212.6.

[14]. Страница инструмента Retrascope. http://forge.ispras.ru/projects/retrascope (дата обращения: 18.07.17)

[15]. Страница библиотеки Fortress. http://forge.ispras.ru/projects/solver-api (дата обращения: 18.07.17)

[16]. Cavada D., Cimatti A., Dorigatti M., Griggio A., Mariotti A., Micheli A., Mover S., Roveri M., Tonetta S. The nuXmv symbolic model checker. Proceedings of the 16th International Conference on Computer Aided Verification (CAV), No.8559, 2014, pp. 334-342. DOI: 10.1007/978-3-319-08867-9_22.

[17]. Страница набора тестов ITC'99. http://www.cad.polito.it/tools/itc99.html (дата обращения: 18.07.17)

[18]. Liu X., Hsiao M.S. On Identifying Functionally Untestable Transition Faults. Proceedings of the Ninth IEEE International High-Level Design Validation and Test Workshop, 2004, pp. 121-126. DOI: 10.1109/hldvt.2004.1431252.

Verification of 10 Gigabit Ethernet controllers

M.V. Petrochenkov <petroch_m@mcst.ru>
R.E. Mushtakov <mushtakov_r@mcst.ru>
I.A. Stotland <stotl_i@mcst.ru>
MCST, 1 Nagatinskaya st., Moscow, 117105, Russia

Abstract. This article proposes approaches used to verify 10 Gigabit Ethernet controllers developed by MCST. We present principles of the device operation – they provide a set of memory-mapped registers and use direct memory access, and their characteristics. We describe a set of approaches used to verify such devices – prototype based verification, system and stand-alone verification. We provide the motivation for the chosen approach – combination of system verification with stand-alone verification of its single component. The structure of the test systems that we used to verify devices and their components are presented. Test system of the controller transmits Ethernet frames to the network and receives frames from it. Algorithms to transfer packet to representation used by the device were implemented. Stand-alone test system was developed for a connector module between internal device buses and its external interface. Test systems were developed using UVM. This methodology and structure of test systems allowed to reuse components in a different systems. A set of test scenarios used to verify the device is described. The examination of network characteristics of the controller is very important in the verification process. Some approaches and techniques for throughput measuring and modes of device operations for the measurement are described. We present measured throughput in different modes. In conclusion, we provide a list of found errors and their distribution by different types of functionality they affected.

Keywords: 10 Gigabit Ethernet; network interface controller; verification; throughput; UVM; test system

DOI: 10.15514/ISPRAS-2017-29(4)-17

For citation: Petrochenkov M. V., Mushtakov R. E., Stotland I. A. Verification of 10 Gigabit Ethernet controllers. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 257-268. DOI: 10.15514/ISPRAS-2017-29(4)-17

1. Introduction

Development of modern computer networks provides the demand for high-speed communication without sacrificing reliability. The evolution of Ethernet standard

(IEEE 802.3 [1]) is an example of ever-rising demand for higher speed networks. Network interface controller (NIC) is the device that connects a computer to the network. Reliability and performance of the controller is very important for organization of modern networks. Network performance and accuracy of its work as a whole depends on the quality of implementation of NICs. To ensure that the controller satisfies all requirements for performance and reliability, it should be thoroughly verified.

Various methods of verification are used at all phases of NIC design flow. Common approaches for the device verification are physical prototype verification, system verification and stand-alone verification.

Process of physical prototype verification uses the device implemented in FPGA as a NIC in a “real” machine. Characteristics of the approach:

- Test stimuli are generated using operating system network drivers and signals from physical network (in our case - third-party 10 Gigabit Ethernet controllers).
- The fastest approach by a wide margin.
- Ability to execute “real life” scenarios and gather information to improve the device performance for most important use cases.
- Ability to debug network drivers.
- Difficulty in localization of detected errors.
- Slow iteration cycle due to slow recompilation to FPGA process.

In the system, verification approach a NIC is simulated as a part of whole System on a Chip (SoC). NIC is configured according to required settings and then it executes network transactions. Characteristics of the system verification approach for the Ethernet controller:

- Test stimuli are memory access operation to the device registers and received Ethernet packets - very similar to typical mode of device operation.
- Simpler localization of detected errors and better error detection tools than using physical prototyping.
- Ability to implement directed scenarios from a physical prototype.
- Difficult and time consuming to cover all possible situations, especially for complex internal components.
- Requires all device components to be in working state.

In stand-alone verification, a single device component is simulated. Typically, it is used for components with (1) high internal complexity and (2) which reliability is crucial for the device [2]. Properties of the approach:

- Test stimuli are transactions on the external interfaces of the component.
- Could be start as soon as only RTL-model of device component is ready (not the whole device).

- Faster simulation for smaller device subset.
- It is easier to create specific test cases and more complex test scenarios for component under test.
- Information about internal interfaces of the device is required.
- Cannot eliminate the need of system verification completely and thus always requires extra labor and other resources.

In our previous projects, we used only physical prototyping and system verification to verify NICs. However, there are types of errors that are hard to find using only these approaches. In this regard, all aforementioned methods we used during verification process of 10 Gigabit Ethernet controllers. Separate team used physical prototype verification approach, and further discussion of it is beyond the scope of this article.

In this paper, we present a case study for functional verification of 10 Gigabit Ethernet controllers developed by MCST. The paper addresses the problem and methods of stand-alone verification of 10 Gigabit Ethernet controllers.

The rest of the paper is organized as follows. In the Section 2, we describe the devices under test: RTL-models of the 10 Gigabit Ethernet controllers, their features and intended methods of implementation. Section 3 presents different test systems developed for components of the controller and the complete controller. In Section 4 we give further insight into the process of examination of the device network properties - most importantly its throughput. Section 5 presents the results of verification and plans of future work.

2. Device Under Test for Different 10 Gigabit Ethernet Controller Implementation

Model of the 10 Gigabit Ethernet Controller is implemented using Verilog Hardware Description Language (HDL). It is RTL (register transfer level) description that is used in different implementation (Device Under Test, DUT):

- FPGA-based network controller (based on Altera Cyclone V [3]). This FPGA provides a set of components that were used in the device: PCI Express Hardware IP module that implements physical and data link layer of the protocol, and a set of configuration space registers, and XAUI Hardware IP module to transform 10 Gigabit Media Independent Interface (XGMII) signals. It is connected to the other parts of the system using standard interface Avalon [4].
- ASIC-based network controller - a part of a currently developed Elbrus-16C System on Chip (SOC). Controller is connected to the rest of the system using in-house interface (SLink) to transfer packets based on PCI Express transaction layer packets.

General schemes of both DUT are presented in figure 1. Both types of the DUT share the Ethernet Control Module and implement the same programming interface. This interface is typical for PCI and PCI-Express devices. A set of memory-mapped registers are used to control device behavior. Those registers can be separated into four groups:

1. PCI Express registers - common set of registers of PCI Express devices. They are used to control access to internal memory of the devices and from the device to system memory. It also implements basic interrupt control.
2. Media Access Control (MAC) registers allow to control the Ethernet physical layer. They are used to control pause frames, control sum (CRC) calculation, non standard-compliant frames and limit the speed of packet transmission.
3. Transceiver (TX) registers control the transmission of packets from the system to the network, calculation of CRCs for higher-level protocols supported by the controller (IPv4 and IPv6, TCP and UDP).
4. Receiver (RX) registers control the reception of packets from the network, packet filtering and control sum checking for IP, TCP/IP and UDP/IP packets.

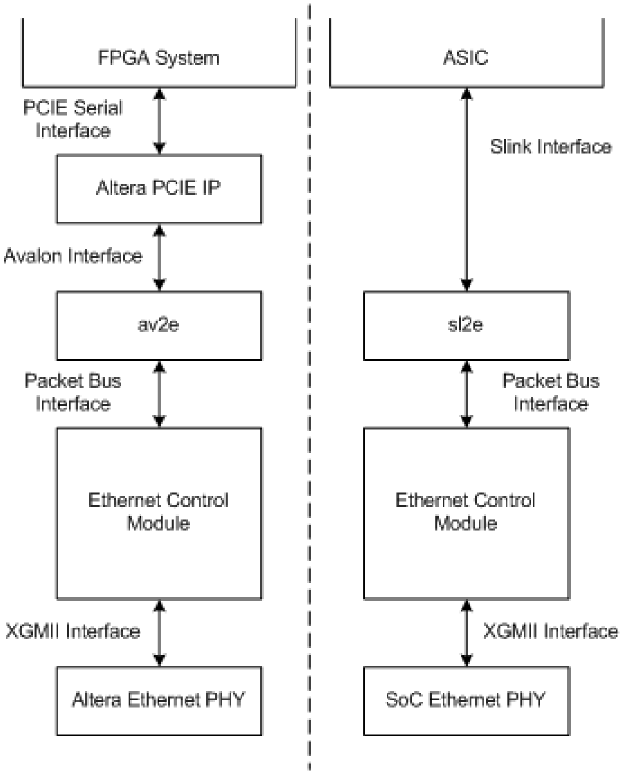


Fig. 1. Devices under test for FPGA and ASIC-based in SoC implementation.

To allow multiple processes to work in parallel with a single controller TX and RX registers contain several identical groups of registers (descriptor queues).

Ethernet Control module uses universal packet bus interface. This interface provides convenient access to large continuous areas of memory where Ethernet packet data are stored. Altera PCI Express and SLink interfaces work with packets of size up to 64 bytes. To increase the rate of data transmission and reduce the CPU involvement into device operation it used direct memory access (DMA). Devices use different modules to connect packet bus to Avalon and SLink interfaces. Internal names of those connector modules are av2e (avalon to everything) and sl2e (SLink to everything). Those connectors implement DMA by splitting packet bus transactions and transforming them into memory access operations.

3. Test Systems for 10 Gigabit Ethernet Controller Verification

As stand-alone verification could be started as soon as RTL-model of device component is ready, without waiting RTL-model of the whole device. Verification of the 10 Gigabit Ethernet Controller process was started at the same time as the development of the FPGA-based controller and system on chip (SoC). This approach allowed identifying errors earlier, and reducing total development time of the device. To check correctness of the controller model, it is included in a test system — a program that generates test stimuli, checks validity of reactions and determines verification quality.

There are several verification methodologies in order to develop constraint-random coverage-driven verification test systems. A verification methodology provides guidelines, class libraries and macros libraries. The Universal Verification Methodology (UVM) [5] is currently the most widespread verification methodology. UVM allows automating test system design process and makes it easier to add new components and collecting the functional coverage [6]. In paper [7] the approach to UVM test system developing for Gigabit Ethernet is presented. However, Gigabit Ethernet has some differences in protocol and interfaces from 10 Gigabit Ethernet. Moreover, in our case we have to verify in-house Slink interface communication.

For stand-alone verification of the 10 Gigabit Ethernet Controller, we developed two stand-alone UVM test systems based on two different DUTs for FPGA and ASIC-based implementation. The structures of the test systems and approaches used to process verification both of DUTs are presented below.

3.1 Test Systems for FPGA-based 10 Gigabit Ethernet Controller Verification

The top-level module of the 10 Gigabit Ethernet Controller is called XGBE (10 GigaBit Ethernet). The structure of the test system for XGBE stand-alone verification is provided in figure 2.

In the controller, packet is represented as one or multiple (split) descriptors and a payload stored in the system memory. Each transmit and receive descriptor queue in the device works with continuous area of memory where descriptors are stored. The controller implements head and tail pointer registers used to determine descriptors currently in use. Each descriptor contains a payload pointer. XGBE driver transforms Ethernet packet between representation used by the controller and representation in the test system - UVM-base transaction class.

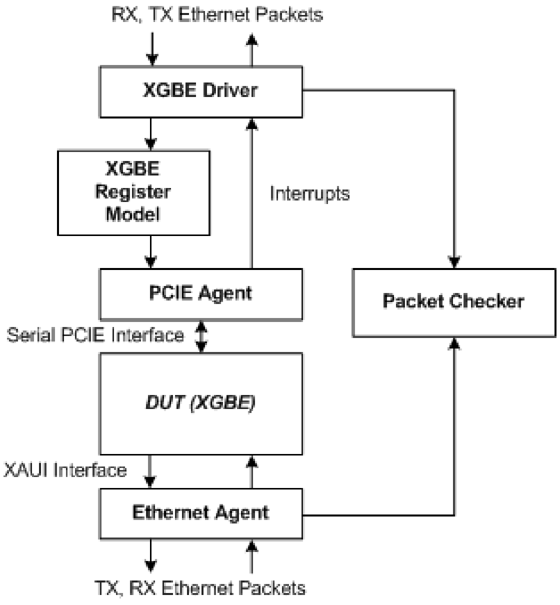


Fig. 2. Structure of XGBE test system.

Algorithm for packet transmission:

- Allocate memory for payload.
- Form corresponding packet descriptor.
- Write descriptor(s) to first free memory location in a queue.
- Change queue head pointer value.
- Wait until tail value becomes equal to head.
- Collect packet transmission information and free used memory resources.

Packet reception algorithm works in a similar way, but because we do not have an information on expected packets sizes, algorithm works in two threads: descriptor preparation and packet reception.

Descriptor preparation works as follows:

- Wait when test system requests additional space for packet reception (conditions of this request are generally test-specific and determined by test system settings).

- Allocate memory for the number of descriptors and fill corresponding descriptor memory.
- Increase RX queue head value.

Packet reception algorithm:

- Wait for change of RX queue tail value.
- Collect received packet data from descriptor and payload.
- Free resources used allocated in descriptor preparation routine.

To simplify access to device registers and abstract away details of register access operations, UVM register model (XGBE Register Model) of the controller was developed. This model uses a bus adapter to transform generic register access operations to the required bus format (in our case - transactions for PCIE agent). Other features of PCIE agent used by device driver are direct access to system memory and interrupt notifications.

Test system was used to verify the device on various test sequences, directed at different device functions. Test can be separated into four large groups: data flow tests, filtering tests, packet parsing tests and throughput tests. The general goal of the first group is to ensure that data processed by the controller will stay correct. At first, maximum possible packet flow through the device was tested. Later we started introducing different bottlenecks (by means of Ethernet Pause frames, PCI Express credits, limiting the size of transmission and reception buffers, available amount of receive descriptors etc...) to achieve different events in the internal components of the controller. The goal of the second group is to ensure correctness of filtering capabilities of device. A set of packets are generated in a way to be test all available packet filters. For the third group, higher-level protocol packets are encapsulated in the basic Ethernet frame and the ability of the device to handle them correctly (packet type detection, automatic checksum calculation etc.) was tested. Throughput tests will be discussed separately later in the article.

It also was decided that the stand-alone verification was necessary for a single type of module in the device - connector between multiple internal packet buses and the external PCI Express like interface (Altera Avalon Interface of PCI Express module), due to several reasons:

- These modules are relatively independent from the rest of the system, its early completion allowed for early verification start.
- High complexity of this module is due to complex rules of transaction splitting.
- Different interactions between all packet bus requesters are difficult to achieve in a complete system.

Its structure is provided in a figure 3. Connector module communicates through Altera PCIE module with the memory inside the PCI Express agent. Information about Ethernet packets is stored in this memory. These data can be separated into two groups: packet descriptors (which are used by the controller to facilitate data transfer) and payload of packets themselves. Connector module solves several problems. First, it transforms requests from packet bus to PCIE memory access transactions. Second, it transforms the responses to those requests, to format convenient for the rest of the devices. It was decided to include third party PCIE module that (we presume) is well-tested and bug-free because we have access to PCI Express agent, but do not have one for the Avalon interface. Additional performance gained through exclusion of this module is compensated by the time and other resources needed for development of Avalon agent.

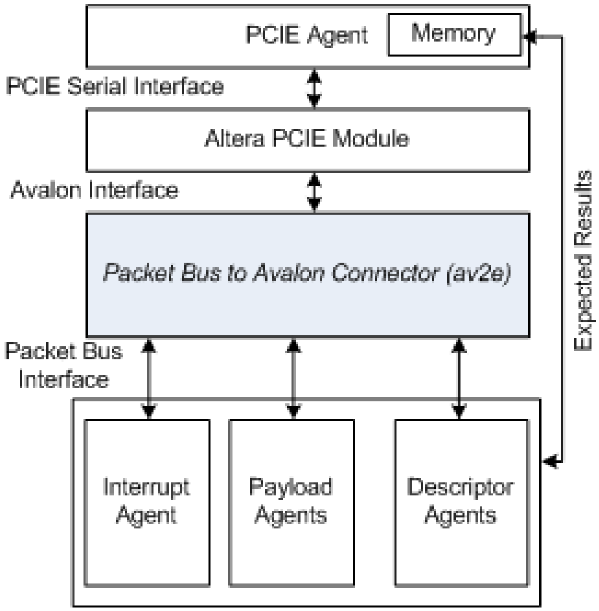


Fig. 3. Structure of av2e module test system.

To verify av2e module (connector to Avalon interface) as a part of 10 Gigabit Ethernet controller a test system was designed. The test system is also designed using UVM and consists of the set of components, which are inherited from standard classes of UVM library.

The exchange between these components is carried out by transactions that facilitates scaling and configurability of system. Developed components could be adapted for use in system for verification of a whole controller to speed up its development.

3.2 Test System for ASIC-based 10 Gigibit Ethernet Controller Verification

Test system for connector between packet buses and system SLink interface module (sl2e) is similar to one used for verification of av2e module. PCI Express agent and

(and Altera PCI Express module) were replaced with SLink agent which provides similar interface for other parts of test system:

- Transformation of transaction-level PCI Express operations into interface signals.
- Access to internal memory of the agent.
- Automatic generation of the completions for upstream requests.
- Notification mechanism for special requests.

The test system for ASIC-based 10 Gigabit Ethernet controller was developed by replacing PCIE Agent with SLink agent.

Usage of various test system components by different test systems is summarized in table 1. Only limited part of device register model (PCI Express Configuration space registers) was actively used in av2e and sl2e test systems.

Table. 1. Test system component reuse.

	FPGA XGBE	av2e	ASIC XGBE	sl2e
Bus Agent	-	+	-	+
PCIE Agent	+	+	-	-
SLink Agent	-	-	+	+
Register Model	+	+/-	+	+/-
XGBE Driver	+	-	+	-

3. Throughput Analyzing

Throughput is one of the most important characteristics of any network controller. In our case, the controller should support throughput of 10 Gigabit per second for packet transmission and reception. Therefore, tests system must support the development of special scenarios for throughput testing. It is implemented in the test system by limiting the test system and device configuration in a way that it will not introduce new “bottleneck”.

To achieve necessary controller throughput it is essential to ensure that every component satisfy the requirement. In the controller, PCI Express Gen2 x4 bus was used as a connection to the system. Maximum possible value of throughput for this bus is 16 Gigabit per second. Thus, this bus satisfies the requirements. Throughput of av2e module was measured after its verification was complete. To do that, the throughput analyzer was developed. It executes all necessary calculations using the information about the start time of first packet`s data transmission and the end time of the last one. Initial value of av2e module was ~11.2 Git/s in both directions. This value is higher than maximum packet flow from the network, so this module will not serve as a “bottleneck”.

Measurements of throughput of whole controller started after the verification of single packet transfer. Separate packet analyzers were designed for transmission to network and reception from it. Principle of measurement is similar to the one described above: analyzer collects the information on transmission start time, end time and size of transferred data. A set of tests were developed to determine throughput for different packet flows: (1) transmission, (2) reception, (3) mixed and (4) loopback. In addition, for each of these tests it is possible to select mode of operation: either raw Ethernet packets of mixed UDP/TCP flow to ensure that packet parsing will not slow down the device.

The first measured value of throughput was 0.36 GBit/s in loopback mode. This, of course, does not satisfy the requirements. After multiple performance improvements, the goal to achieve maximum possible throughput for separate transmission and reception packet flow was achieved. At this moment, throughput value for mixed mode 10Gbit/s for reception and 4GBit/s for transmission and 7 GBit/s for both values in loopback mode. Work to further improve the performance is ongoing.

4. Results

To verify the 10 Gigabit Ethernet Controllers four separate test systems were designed using a set of components. Test sequences were developed to test the correctness of the device for each test system.

A total of 49 test scenarios were used to verify all functions of av2e module: read and write operations with different parameters, sequentially and in parallel. Total number of bugs detected by the av2e test system is 14. Found errors are the corruption of transmitted or received data and complete loss of packets by the components.

Number of test for the whole 10 Gigabit Ethernet controller is 31.They thoroughly check that the device works as described in the specification. Different test scenarios check different modes of operation: transmission of packets from system to network, reception of packets from network, mixed flow and loopback mode. The check proper handling of different types of payload (Raw Ethernet, or encapsulated IPv4, IPv6, UDP, TCP, Runt Frames, PTP packets), working with packets with different priorities, working with packets with vlan tags, pause frames, checking of packet filtering capabilities and automatic calculation of checksums. Different ways of interaction with the system are also checked: correctness of interrupts and mirroring of certain device registers in memory.

As a result of verification of the controller, 74 errors were discovered and corrected. Those can be divided into 3 groups:

- Errors in data transmission is the biggest group of 49 errors. Those errors caused the transmission of incorrect data in packets by the controller. This group includes such errors as: partial loss of data, duplication of received data, “merging” of different packets into one and incorrect calculation of checksum.
- Number of errors in packets parsing and filtering is 9. They caused incorrect detection of types of packets, incorrect placement of CRCs in packet and incorrect filtering of received packets.

- Group of 16 miscellaneous errors have not caused errors in packet transfer. Those errors appeared during accessing internal registers of the device or caused suboptimal utilization of the device resources.

All above-mentioned errors were corrected. ASIC-based version of the device and its test system are currently under active development. Our future works is aimed at further verification of ASIC-based version of the 10 Gigabit Ethernet Controller, developing UVM-based reusable components (UVC) for PCIE, Avalon, Slink interfaces for using in test system for other network controllers.

References

[1]. IEEE Standard for Ethernet. IEEE Std 802.3-2012. 1983 p.
[2]. Petrochenkov M., Stotland I., Mushtakov R. Approaches to Stand-alone Verification of Multicore Microprocessor Caches. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 3, pp. 161-172. DOI: 10.15514/ISPRAS-2016-28(3)-10
[3]. Cyclon V – Overview. URL: https://www.altera.com/products/fpga/cyclone-series/cyclone-v/overview.html (accessed 09.04.2017).
[4]. Avalon Interface Specification. Altera. MNL-AVABUSREF. 2015.12.10. 101 Innovation Drive. San Jose, CA 95134. URL: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/mnl_avalon_spec.pdf (accessed 09.04.2017).
[5]. Standard Universal Verification Methodology. URL: http://accellera.org/downloads/standards/uvm (accessed 09.04.2017).
[6]. Stotland I., Shpagilev D., Petrochenkov M. Osobennosti funkcional'noj verifikacii kontrollerov vysokoskorostnyh kanalov obmena mikroprocessornyh sistem semejstva "Elbrus" [Features of High Speed Communication Controllers Standalone Verification of “Elbrus” Microprocessor Systems]. Voprosy radioelektroniki, seriya EVT [Issues of Radioelectronics, the series EVT], 2017, 3, pp. 69-75.
[7]. S. Chitti, P. Chandrasekhar, M. Asha Rani. “Gigabit Ethernet Verification using Efficient Verification Methodology”. Proc. of International Conference on Industrial Instruments and Control (ICIC), College of Enginnering Pune, India. May 28-30, 2015, pp.1231-1235.

Верификация контроллеров 10 гигабитного Ethernet

М.В. Петроченков <petroch_m@mcst.ru>
Р.Е. Муштаков <mushtakov_r@mcst.ru>
И.А. Стотланд <stotl_i@mcst.ru>
АО «МЦСТ», 117105, Россия, г. Москва, ул. Нагатинская, д.1, стр.1

Аннотация. В статье приведены подходы, использовавшиеся в процессе верификации контроллеров 10 гигабитного Ethernet, разработанных в АО «МЦСТ». Описаны принципы работы устройств – они предоставляют программисту набор регистров, отображаемых в память, а также используют прямой доступ к памяти. Представлен набор подходов, применяемых при верификации подобных устройств – верификация физического прототипа, системная и автономная верификация. Описана мотивация выбора подхода – комбинации системной верификации целого устройства и автономной верификации одного из его компонентов. В статье дано описание тестовых систем, использовавшихся для верификации устройств. Тестовая система всего устройства осуществляет передачу Ethernet пакетов в сеть и их прием из сети. Разработаны и описаны алгоритмы преобразования пакетов в представление, используемое устройством. Для модулей связи между внутренними пакетными шинами и внешним интерфейсом была разработана автономная тестовая система. При разработке тестовых систем использовалась методология UVM. Выбранная методология, а также предложенные структуры тестовых систем позволили использовать одни и те же компоненты в различных тестовых системах. Приведен набор тестовых сценариев, разработанных для тестовых систем. Особо важным является процесс верификации пропускной способности устройства. Описаны методы, использовавшиеся для измерения пропускной способности устройства, а также режимы работы контроллера, в которых проводилось измерение. Представлены текущие значения пропускной способности устройства, которых удалось достигнуть в различных режимах. В заключение приведен список найденных ошибок, а также описаны те функции устройства, на которые они влияли, а также направления дальнейшей работы.

Ключевые слова: 10 гигабитный Ethernet; контроллер сетевых интерфейсов; верификация; пропускная способность; UVM; тестовая система

DOI: 10.15514/ISPRAS-2017-29(4)-17

Для цитирования: Петроченков М.В., Муштаков Р.Е., Стотланд И.А. Верификация контроллеров 10 гигабитного Ethernet. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 257-268 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-17

Список литературы

[1]. IEEE Standard for Ethernet. IEEE Std 802.3-2012. 1983 p.

[2]. Petrochenkov M., Stotland I., Mushtakov R. Approaches to Stand-alone Verification of Multicore Microprocessor Caches. Trudy ISP RAN, vol. 28, 3, pp. 161-172. DOI: 10.15514/ISPRAS-2016-28(3)-10

[3]. Cyclon V – Overview. URL: https://www.altera.com/products/fpga/cyclone-series/cyclone-v/overview.html (дата обращения 09.04.2017).

[4]. Avalon Interface Specification. Altera. MNL-AVABUSREF. 2015.12.10. 101 Innovation Drive. San Jose, CA 95134. URL: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/mnl_avalon_spec.pdf (дата обращения 09.04.2017).

[5]. Standard Universal Verification Methodology. URL: http://accellera.org/downloads/standards/uvm (дата обращения 09.04.2017).

[6]. Стотланд И.А., Шпагилев Д.И., Петроченков М.В. Особенности функциональной верификации контроллеров высокоскоростных каналов обмена микропроцессорных систем семейства «Эльбрус». Вопросы радиоэлектроники, серия ЭВТ, 2017, 3, стр. 69-75.

[7]. S. Chitti, P. Chandrasekhar, M. Asha Rani. “Gigabit Ethernet Verification using Efficient Verification Methodology”. Proc. of International Conference on Industrial Instruments and Control (ICIC), College of Enginnering Pune, India. May 28-30, 2015, pp.1231-1235.

Creating Test Data for Market Surveillance Systems with Embedded Machine Learning Algorithms

O. Moskaleva <olga.moskaleva@exactprosystems.com>
A. Gromova <anna.gromova@exactprosystems.com>
Exactpro, LSEG,
20A Building 4, 2nd Yuzhnoportovy Proezd, Moscow, 115088, Russia

Abstract. Market surveillance systems, used for monitoring and analysis of all transactions in the financial market, have gained importance since the latest financial crisis. Such systems are designed to detect market abuse behavior and prevent it. The latest approach to the development of such systems is to use machine learning methods that largely improve the accuracy of market abuse predictions. These intelligent market surveillance systems are based on data mining methods, which build their own dependencies between the variables. It makes the application of standard user-logic-based testing methodologies difficult. Therefore, in the context of intelligent surveillance systems, we built our own model for classifying the transactions. To test it, it is important to be able to create a set of test cases that will generate obvious and predictable output. We propose scenarios that allow to test the model more thoroughly, compared to the standard testing methods. These scenarios consist of several types of test cases which are based on the equivalence classes methodology. The division into equivalence classes is performed after the analysis of the real data used by real surveillance systems. We tested the created model and discovered how this approach allows to define its weaknesses. This paper describes our findings from using this method to test a market surveillance system that is based on machine learning techniques**.**

Keywords: test data; equivalence classes; market surveillance systems; machine learning

DOI: 10.15514/ISPRAS-2017-29(4)-18

For citation: Moskaleva O., Gromova A. Creating Test Data for Market Surveillance Systems with Embedded Machine Learning Algorithms. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 269-282. DOI: 10.15514/ISPRAS-2017-29(4)-18

1. Introduction

1.1 Market surveillance systems

Electronic trading platforms have become an increasingly important part of the financial market in recent years. They are obligated to take legal responsibilities [1], [2] and correspond to the law and the regulatory requirements. Therefore, all market events in the contemporary electronic trading platforms are monitored and analysed by market surveillance systems.

Such systems are designed to detect market abuse behavior and prevent it. Their main goals are detection and prevention of such market abuse cases as insider trading, intentional and aggressive price positioning, creation of fictitious liquidity, money laundering, marking the close, etc. [3]. Different data mining methods are used for improving the quality of the surveillance systems' work [4], [5], [6], [7], [8], [9].

1.2 Quality assurance for market surveillance

The standard quality assurance (QA) methods and technologies seem to be powerless in regard to machine learning (ML) applications. C. Murphy, G. Kaiser, M. Arias even introduced a concept of "non-testable" applications [10]. From the QA perspective, we do not have to test whether an ML algorithm is learning well, but to ensure that the application uses the algorithm correctly, implements the specification and meets the users' expectations. In this paper, we employ the term "testing" in accordance with the QA theory.

It is clear that a sufficient input data set is needed for high-quality testing coverage. Furthermore, the testing data set should be as close as possible to the real data or should even be real. So, which approach should be used for creating data to verify the implementation of an ML-algorithm more fully?

We can test a market surveillance system in the following ways:

- by creating test cases which are based on the knowledge of the business rules from the specification. Such test data are similar to the real users' behaviour;
- by generating various datasets which contain different combinations of variable.

Both variants are suitable for the surveillance systems that use standard control flows, like loops or choices. For standard systems, there is a set of rules, which allows getting a clear output result for specific input data. When it comes to intelligence systems, it is not normally obvious what will happen as a result of certain input because an ML algorithm builds its own dependencies between the variables and human interpretation of such dependencies is impossible. Because of this, it is important to be able to create a set of test cases that will generate obvious and predictable output. Therefore, the second approach to generating the test data allows for the creation of output that is easily interpretable.

1.3 Contribution

This paper introduces the following contribution:

- creating a model for classifying the transactions. This model can be used for the detection of market manipulations;
- test cases generation for defining the weaknesses of the created model. The test cases are based on the equivalence classes;
- testing the prototype based on the created model and the analysis of the received results.

2. Related work

2.1 Ongoing problems in the quality assurance of the modern surveillance systems

It is known that the system containing ML algorithms should learn using a dataset that is real or close to real. Obviously, for testing purposes, it is necessary to use a dataset with a similar structure.

Moreover, during the creation of this dataset, it will be helpful to emphasize the variability of values of the attributes included in the sampling. By generating a variety of combinations with different values, we will create test cases to find out the weaknesses in the ML algorithm, which are related to the separation of classes.

2.2 Existing approaches

C. Murphy, G. Kaiser, M. Arias proposed to divide the data into equivalence classes to generate test cases for testing the "non-testable" software [10]. It should be noted that forming equivalence classes is a standard approach in quality assurance [11].

However, C. Murphy, G. Kaiser, M. Arias suggest to follow three steps in testing this kind of software. Firstly, the data should be divided into several classes, taking into account the size of the dataset, the potential ranges of the attributes and the label values, etc. Then, the test cases, that are based on the investigation of the ML algorithm used in this system, should be created. At last, several testing datasets should be generated.

Supposing that a dataset for QA purposes has been defined, based on the knowledge of the method used in the machine learning system. For solving the problem of the dataset sufficiency, C. Murphy, G. Kaiser, M. Arias propose a “parameterized random data generation” methodology. This methodology enables us to generate large datasets and randomly control them [12], [13], [14].

In their paper, J. Zhang et al. suggest to use predictive mutation testing, as it allows to make decisions without executing the costly mutation testing [15].

Thus, it becomes clear that new methodologies for testing ML applications are being developed. However, the contemporary market surveillance systems impose additional requirements on them. These systems should be able to self-detect the incorrect behavior and classify alerts, and further point at the initiator of this behavior. This should always be noted during the process of creation of test scenarios.

3. Definitions and assumptions

3.1 Structure of the transaction log

A transaction is an event that happens on the financial market and changes any financial instrument, for example:

- submitting a buy-order for security S with price P and volume V;
- cancelling an order with id I;
- trading on security S at price P with volume V, etc.

Transaction logs are the data that are used by the ML surveillance system. Each transaction is stored as an object and has a set of input parameters (transaction characteristics) and one output parameter (presence or absence of the alert):

$I = \{i_1, i_2, \dots, i_j, \dots, i_n\}$, where $i = \overline{\{1, n\}}$;

$i_j = \{TID, B, IID, Side, CP, ExP, ExS, TV, S, TInt, Alert\}$;

Where:

$TID \in N$,

$B = \{Broker_1, Broker_2, \dots, Broker_k\}$, where k is the number of brokers that are available in the configuration file,

$IID = \{Instrument_1, Instrument_2, \dots, Instrument_m\}$,where m is the number of instruments that are available in the configuration file,

$Side = \{Buy, Sell\}$,

$CP \in Q$,

$ExP \in Q$ and $ExP \geq 0$,

$ExS \in Z$,

$TV \in Z$,

$S = \{OAC, RT, CAC, ReOAC, ResumeAC, Halt\}$,

$TInt \in N$,

$Alert = \{0,1\}^M$.

For more attribute details, please refer to Table 1.

Table 1. Attribute details

Parameter	Type	Comment

Transaction ID	numerical	Unique identifier of transaction
Broker	categorical	Company to which the trader belongs
Instrument ID	categorical	Identifier of the instrument to be traded
Side	categorical	Side of the order
ChangePrice	numerical	$ChP = LastTradedPrice - CurrentPrice$
Executed Price	numerical	Price of the trade
Executed Size	numerical	Volume of the Trade
Total Volume	numerical	Total volume traded on the instrument
Session	categorical	Current session on the instrument
Time Interval	interval	Time interval between transactions
Alert	boolean	The output: 0 - regular transaction,1 - suspicious transaction

Each transaction can cause several different alerts, that is why every alert should be classified. This type of classification is called a classification with overlapping classes.

3.2 Market abuse alert

The following type of behavior can be considered as abusive: a broker tries to raise or bring down the price on several trades. The alert will be triggered if the price deviation and the traded volume reach the threshold values. If the price goes up, the buy-orders should be checked, if the price goes down – the sell ones.

4. Background

As each transaction is stored as an object and has a set of independent variables and one dependent variable, for testing purposes, these data should be divided into several equivalence classes. All the objects in one equivalence class have the same characteristics of certain attributes [11] and trigger the same system behavior.

It is important to analyze the transactions, the system behavior and the meaningful parameters before forming the classes. We can use the following types of test cases which are based on equivalence classes. These classes are defined after the analysis of real data used by the surveillance systems:

1) **Consistency checks** with consideration of categorical transaction attributes ($Firm$, $Session$, $Action$, etc.):

   a) Let $a = a_1$ and $class = 1$, and $a = a_2$ and $class = 0$. The categorical attribute gets a concrete value for several test cases, but the other transaction attributes have different values. For such combinations, the $class$ value is set to 1. This is a check for how strongly the $class$ value correlates with the concrete value of the I-th attribute.

b) Let a categorical attribute get any values, always leaving the $class$ set to 1. Let attribute Let $b = b_1$ or $b = b_2$ or $b = b_3$, $class = 1$. It checks that the system can assume that this attribute is inessential.

c) Let $0 \leq c \leq 10$, $class = 1$. The same as for (b), but for numeric variables.

d) Let $c \geq 0$, $class = 1$. The same as for (b), but for numeric variables.

2) **Checks for empty values.** Due to the fact that a surveillance system analyses all the transactions on the financial market and that the considered transaction can be different for each type of alert, it is possible that some values will be empty.

3) **Checks for the presence of noise.** We perform checks for the presence of noise using the equivalence classes for numeric transaction attributes. Some attributes can have values in the concrete range, but mostly they take the average value. For example, the price percentage variable takes the values from 0 to 100 with the mean of 50 and has a low variance. Even if the border values (0 and 100) are included in the acceptable range, they appear so rarely that we treat them as frequency emissions.

5. Approach

5.1 Classification model of market abuse

To prove that the proposed methodology is effective, we have created a model which allows to classify the transactions by one type of abusive behavior. It should be noted that we have to deal with rather specific data, i.e. financial transactions.

It is important to make a thorough analysis of the data related to the business logic, the system requirements and the structure of transaction logs, to be able to define the attributes and their particular qualities. The correct outcomes can only be provided when considering all these factors. Thus, this particular dataset structure, as it is presented in Table 1, was selected for this type of abusive behavior.

We extracted 639 transactions from the messages of one of the electronic trading protocols and manually classified them, using these data to build the classifier. We used a decision tree algorithm for our research. After that, the classifier was trained on a set and its performance was evaluated.

Precision, recall, and F-measure are the evaluation metrics that we used for our calculations. Values of these metrics that we received are represented in Table 2.

Table 2. Values of metrics

Precision	Recall	F-measure
0.615	0.678	0.645

Table 3. Detailed examples of test cases

Scenario 1: Consistency checks 1.a. Instrument		
Test case 1	Test case 2	Test case 3
$B = Broker_1$ $IID = Instrument_1$ $Side = Buy$ $CP = CP_1$ $ExP = P_1$ $ExS = S_1$ $TV = TV_1$ $S = RT$ $TInt = TI_1$ $Alert = 1$	$B = Broker_2$ $IID = Instrument_1$ $Side = Sell$ $CP = CP_1 - 1$ $ExP = P_2$ $ExS = S_2$ $TV = TV_2$ $S = RT$ $TInt = TI_2$ $Alert = 1$	$B = Broker_3$ $IID = Instrument_1$ $Side = Sell$ $CP = CP_1 + 2$ $ExP = P_3$ $ExS = S_3$ $TV = TV_3$ $S = RT$ $TInt = TI_3$ $Alert = 1$
Scenario 2: Consistency checks 1.b. Broker		
Test case 1	Test case 2	Test case 3
$B = Broker_2$ $IID = Instrument_1$ $Side = Buy$ $CP = CP_1$ $ExP = P_1$ $ExS = S_1$ $TV = TV_1$ $S = RT$ $TInt = TI_1$ $Alert = 1$	$B = Broker_3$ $IID = Instrument_1$ $Side = Buy$ $CP = CP_1$ $ExP = P_1$ $ExS = S_1$ $TV = TV_1$ $S = RT$ $TInt = TI_1$ $Alert = 1$	$B = Broker_1$ $IID = Instrument_1$ $Side = Buy$ $CP = CP_1$ $ExP = P_1$ $ExS = S_1$ $TV = TV_1$ $S = RT$ $TInt = TI_1$ $Alert = 1$
Scenario 3: Consistency checks 1.c. Executed Price		
Test case 1	Test case 2	Test case 3
$B = Broker_2$ $IID = Instrument_1$ $Side = Buy$ $CP = CP_1$	$B = Broker_2$ $IID = Instrument_1$ $Side = Buy$ $CP = CP_1$	$B = Broker_2$ $IID = Instrument_1$ $Side = Buy$ $CP = CP_1$

$ExP = P_1$ $ExS = S_1$ $TV = TV_1$ $S = RT$ $TInt = TI_1$ $Alert = 1$	$ExP = P_2$ $ExS = S_1$ $TV = TV_1$ $S = RT$ $TInt = TI_1$ $Alert = 1$	$ExP = P_3$ $ExS = S_1$ $TV = TV_1$ $S = RT$ $TInt = TI_1$ $Alert = 1$
Scenario 4: Consistency checks 1.d. Total Volume		
Test case 1	Test case 2	Test case 3
$B = Broker_2$ $IID = Instrument_1$ $Side = Buy$ $CP = CP_1$ $ExP = P_1$ $ExS = S_1$ $TV = TV_1$ $S = RT$ $TInt = TI_1$ $Alert = 1$	$B = Broker_2$ $IID = Instrument_1$ $Side = Buy$ $CP = CP_1$ $ExP = P_1$ $ExS = S_1$ $TV = TV_2$ $S = RT$ $TInt = TI_1$ $Alert = 1$	$B = Broker_2$ $IID = Instrument_1$ $Side = Buy$ $CP = CP_1$ $ExP = P_1$ $ExS = S_1$ $TV = TV_3$ $S = RT$ $TInt = TI_1$ $Alert = 1$
Scenario 5: Checks for empty values. Executed Price		
Test case 1	Test case 2	Test case 3
$B = Broker_2$ $IID = Instrument_1$ $Side = Buy$ $CP = CP_1$ $ExP = empty$ $ExS = S_1$ $TV = TV_1$ $S = RT$ $TInt = TI_1$ $Alert = 1$	$B = Broker_3$ $IID = Instrument_2$ $Side = Sell$ $CP = CP_1$ $ExP = empty$ $ExS = S_2$ $TV = TV_2$ $S = RT$ $TInt = TI_1$ $Alert = 1$	$B = Broker_1$ $IID = Instrument_3$ $Side = Buy$ $CP = CP_1$ $ExP = empty$ $ExS = S_3$ $TV = TV_4$ $S = RT$ $TInt = TI_1$ $Alert = 1$
Scenario 6: Checks for the presence of noise. Executed Size.		
Test case 1	Test case 2	Test case 3
$B = Broker_2$ $IID = Instrument_1$ $Side = Buy$ $CP = CP_1$ $ExP = P_1$ $ExS = S_1$	$B = Broker_2$ $IID = Instrument_1$ $Side = Buy$ $CP = CP_1$ $ExP = P_1$ $ExS = S_2$	$B = Broker_2$ $IID = Instrument_1$ $Side = Buy$ $CP = CP_1$ $ExP = P_1$ $ExS = S_2$

$TV = TV_1$ $S = RT$ $TInt = TI_1$ $Alert = 1$	$TV = TV_1$ $S = RT$ $TInt = TI_1$ $Alert = 1$	$TV = TV_1$ $S = RT$ $TInt = TI_1$ $Alert = 1$

5.2 Prototype Testing

To test our prototype, we have created a dataset with the same structure as the training dataset and performed all the validations described in Section 4. For each type of the validation, we created the test cases based on the equivalence classes. Please refer to Table 3 for detailed examples.

Figure 1 illustrates the proposed approach. Test Data is the whole set of data that will be used to test ML-based surveillance system. Then these data are divided into equivalence classes, as proposed in Section 4. The generated test cases are used for testing Market Surveillance Systems based on ML.

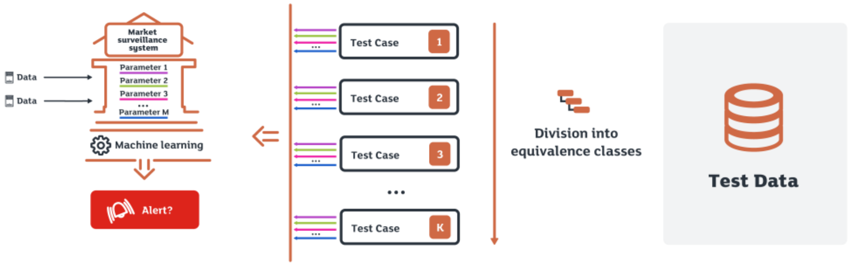


Fig. 1. Prototype Testing

6. Results

After testing our prototype, we received the values of the evaluation metrics that are presented in Table 4.

Table 4. Values of the evaluation metrics

Precision	Recall	F-measure
1.000	0.662	0.797

After a detailed analysis of the instances where the model detected an error, it was observed that errors occurred in:

- different Brokers - Scenario 2: Consistency checks;
- empty values - Scenario 5: Checks for empty values;
- border values - Scenario 6: Checks for the presence of noise.

- Thus, the test that we performed revealed some weaknesses in the ML algorithm:
- An error occurred in one of the categorical attributes during the consistency checks. It may indicate that the model does not take into account the categorical attribute in the algorithm.
- Classification errors occurred in empty values, so the algorithm may miss some abnormal behavior when encountering them. For example, all manipulations at the start of the day may be missed by the surveillance system.
- Some errors were detected in the border values used for the Executed Size variable. According to the general quality assurance theory, we should validate the border values for numeric variables. But in our experiment, unlike what was expected, we saw that the algorithm failed to validate such test cases. The reason is possibly linked with the distribution of the training sample.
- Errors in the tests took place due to thoughtless randomization. After analyzing the results, we can conclude that it is crucial to randomise the dataset with the business logic in mind.

According to these results, it is clear that some details of the dataset and the model need to be considered in the future work.

7. Conclusions and future work

This paper presents the following conclusions:

- The analysis of transactions extracted from an electronic trading protocol allowed us to successfully create a model for the classification of market abuse behavior.
- The proposed scenarios allowed to test the model more thoroughly. The following checks were included: consistency checks, checks for border values, checks for empty values, etc.
- The prototype testing revealed some weaknesses that manifested themselves through unexpected behavior in the case of empty values, in the case of border values (for some of the attributes) and also in the case of different values for one of the categorical attributes.

As focus of our future research, we propose to generate the test cases using the equivalence classes methodology. We advise that the equivalence classes be set according to the types of data (categorical, numeric, etc.) and their border values. Such an approach enables us to parameterize the equivalence classes and to further develop an automatic tool that generates the test data.

References

[1]. FCA (financial conduct authority) (online). Available at: https://handbook.fca.org.uk/

[2]. SEC (Securities and Exchange Commission) (online). Available at: https://www.sec.gov/

[3]. FINMAR Financial Stability and Market Confidence Sourcebook (online publication). Available at: https://handbook.fca.org.uk/handbook/FINMAR/

[4]. Cao L., Ou Y., Yu P.: Detecting Abnormal Coupled Sequences and Sequence Changes in Group-based Manipulative Trading Behaviors. In Proc. of KDD'10, Washington, DC, USA, July 25–28, 2010, pp. 85-93

[5]. Donoho S.: Early Detection of Insider Trading in Option Markets In Proc. of KDD'04, Seattle, Washington, USA, August 22–25, 2004, pp. 420-429

[6]. Luo C., Zhao Y., Cao L., Ou Y., Zhang C.: Exception Mining on Multiple Time Series in Stock Market. In Proc. of International Conference on Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM, 2008, pp. 690-693

[7]. Nasdaq and Digital Reasoning Establish Exclusive Alliance to Deliver Holistic Next Generation Surveillance and Monitoring Technology (online publication). Available at: http://www.digitalreasoning.com/buzz/nasdaq-and-digital-reasoning-establish-exclusive-alliance-to-deliver-holistic-next-generation-surveillance-and-monitoring-technology.1884035, 23.02.2016

[8]. Ou Y., Cao L., Luo C., Liu L.: Mining Exceptional Activity Patterns in Microstructure Data. In Proc. of International Conference on Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM, 2008, pp. 884-887

[9]. Ou Y., Cao L., Yu T., Zhang C.: Detecting Turning Points of Trading Price and Return Volatility for Market Surveillance Agents. In Proc. of International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops, IEEE/WIC/ACM, 2007, pp. 491-494

[10]. Murphy C., Kaiser G., Arias M.: An Approach to Software Testing of Machine Learning Applications. Proc of the 19th International Conference on Software Engineering and Knowledge Engineering (SEKE), Boston MA, Jul 2007, pp. 167-172

[11]. Nautiyal L, Preeti: A Novel Approach of Equivalence Class Partitioning for Numerical Input. ACM SIGSOFT Software Engineering Notes. Volume 41 Issue 1, 2016, pp. 1-5

[12]. Murphy C., Kaiser G., Arias M.: Parameterizing Random Test Data According to Equivalence Classes. Proc of the 2nd International Workshop on Random Testing (RT'07), Atlanta GA, Nov 2007, pp. 38-41

[13]. Murphy C., Kaiser G., Arias M.: A Framework for Quality Assurance of Machine Learning Applications. Columbia University Computer Science Technical Reports, New York, 2006

[14]. Murphy C., Kaiser G., Hu L., Wu L.: Properties of Machine Learning Applications for Use in Metamorphic Testing. Proc of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE), Redwood City CA, Jul 2008, pp. 867-872.

[15]. Zhang J., Wang Z., Zhang L., Hao D., Zang L., Cheng S., Zhang Lu.: Predictive Mutation Testing. In Proc. of ISSTA'16, Saarbrücken, Germany, July 18–20, 2016, pp. 342-353

Создание тестовых данных для систем контроля и мониторинга рынка, содержащих встроенные алгоритмы машинного обучения

О. Москалёва <olga.moskaleva@exactprosystems.com>
А. Громова <anna.gromova@exactprosystems.com>
Exactpro, LSEG,
115088, Россия, Москва, 2-й Южнопортовый проезд, 20А, с4

Аннотация. Для правильной обработки информации о возможных сделках, проходящих через торговые платформы, выявления и предупреждения финансовых манипуляций, биржи устанавливают системы контроля и мониторинга данных. Эти системы получили широкое распространение за последние годы. Объемы и темпы торговли постоянно возрастают, увеличивается число разновидностей ценных бумаг. Финансовые регуляторы предъявляют все новые требования к торговым платформам. Из вышесказанного следует, что современные финансовые информационные системы в ближайшие годы будут продолжать совершенствоваться и активно использовать средства машинного обучения. Поэтому в последние несколько лет системы контроля и мониторинга рынка начинают внедрять модули интеллектуального анализа транзакций. Таким образом, интеллектуальные системы контроля и мониторинга рынка требуют усовершенствования подходов к их тестированию. Это связано с тем, что методы интеллектуального анализа данных формируют свои собственные зависимости между переменными. Тестовые сценарии должны разрабатываться таким образом, чтобы ожидаемый результат был понятен и предсказуем. Очевидно, что стандартные методы тестирования требуют модернизации. В представленной статье рассмотрены особенности современных интеллектуальных информационных систем, а также особенности их тестирования. В данном исследовании был разработан прототип модуля классификации финансовых манипуляций. Также предложены тестовые сценарии, позволяющие тестировать разработанный прототип. Данные сценарии состоят из нескольких типов, основанных на методологии классов эквивалентности. Разделение на классы эквивалентности было выполнено после анализа реальных данных. Прототип был протестирован с помощью выше обозначенных сценариев. Предложенный метод позволил выявить недостатки модуля классификации финансовых манипуляций.

Ключевые слова: тестовые данные; классы эквивалентности; системы контроля и мониторинга рынка; машинное обучение.

DOI: 10.15514/ISPRAS-2017-29(4)-18

Для цитирования: Москалёва О., Громова А. Создание тестовых данных для систем контроля и мониторинга рынка, содержащих встроенные алгоритмы машинного обучения. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 269-282 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-18

Список литературы

[1]. FCA (financial conduct authority) (online). Доступно по ссылке: https://handbook.fca.org.uk/

[2]. SEC (Securities and Exchange Commission) (online). Доступно по ссылке: https://www.sec.gov/

[3]. FINMAR Financial Stability and Market Confidence Sourcebook (online publication). Доступно по ссылке: https://handbook.fca.org.uk/handbook/FINMAR/

[4]. Cao L., Ou Y., Yu P.: Detecting Abnormal Coupled Sequences and Sequence Changes in Group-based Manipulative Trading Behaviors. In Proc. of KDD'10, Washington, DC, USA, July 25–28, 2010, pp. 85-93

[5]. Donoho S.: Early Detection of Insider Trading in Option Markets In Proc. of KDD'04, Seattle, Washington, USA, August 22–25, 2004, pp. 420-429

[6]. Luo C., Zhao Y., Cao L., Ou Y., Zhang C.: Exception Mining on Multiple Time Series in Stock Market. In Proc. of International Conference on Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM, 2008, pp. 690-693

[7]. Nasdaq and Digital Reasoning Establish Exclusive Alliance to Deliver Holistic Next Generation Surveillance and Monitoring Technology (online publication). Доступно по ссылке: http://www.digitalreasoning.com/buzz/nasdaq-and-digital-reasoning-establish-exclusive-alliance-to-deliver-holistic-next-generation-surveillance-and-monitoring-technology.1884035, 23.02.2016

[8]. Ou Y., Cao L., Luo C., Liu L.:Mining Exceptional Activity Patterns in Microstructure Data. In Proc. of International Conference on Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM, 2008, pp. 884-887

[9]. Ou Y., Cao L., Yu T., Zhang C.:Detecting Turning Points of Trading Price and Return Volatility for Market Surveillance Agents. In Proc. of International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops, IEEE/WIC/ACM, 2007, pp. 491-494

[10]. Murphy C., Kaiser G., Arias M.: An Approach to Software Testing of Machine Learning Applications. Proc of the 19th International Conference on Software Engineering and Knowledge Engineering (SEKE), Boston MA, Jul 2007, pp. 167-172

[11]. Nautiyal L, Preeti: A Novel Approach of Equivalence Class Partitioning for Numerical Input. ACM SIGSOFT Software Engineering Notes. Volume 41 Issue 1, 2016, pp. 1-5

[12]. Murphy C., Kaiser G., Arias M.: Parameterizing Random Test Data According to Equivalence Classes. Proc of the 2nd International Workshop on Random Testing (RT'07), Atlanta GA, Nov 2007, pp. 38-41

[13]. Murphy C., Kaiser G., Arias M.: A Framework for Quality Assurance of Machine Learning Applications. Columbia University Computer Science Technical Reports, New York, 2006

[14]. Murphy C., Kaiser G., Hu L., Wu L.: Properties of Machine Learning Applications for Use in Metamorphic Testing. Proc of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE), Redwood City CA, Jul 2008, pp. 867-872.

[15]. Zhang J., Wang Z., Zhang L., Hao D., Zang L., Cheng S., Zhang Lu.: Predictive Mutation Testing. In Proc. of ISSTA'16, Saarbrücken, Germany, July 18–20, 2016, pp. 342-353

Using modularization in embedded OS

[1,2] *K.A. Mallachiev <mallachiev@ispras.ru>*
[1,2,3] *N.V. Pakulin <npak@ispras.ru>*
[1,2,3,4] *A.V. Khoroshilov <khoroshilov@ispras.ru>*
[1] *D.V. Buzdalov <buzdalov@ispras.ru>*
[1] *Institute for System Programming of the RAS,*
25, Alexander Solzhenitsyn Str., Moscow, 109004, Russia.
[2] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia.
[3] *Moscow Institute of Physics and Technology (State University)*
9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia
[4] *National Research University Higher School of Economics (HSE)*
11 Myasnitskaya Ulitsa, Moscow, 101000, Russia

Abstract. Modern embedded OS are designed to be used in control solutions in various hardware contexts. Control computers may differ in the architecture of the CPU, the structure of communication channels, supported communication protocols, etc. Embedded OS are often statically configured to create an OS image, which intended to be executed on some specific control computer. System integrator usually performs this configuration. Embedded OS are often developed by many companies. Joint development and integration is very complex if OS doesn’t support modularity. Support of modularity and component assembly reduces the need of communication among companies during development and integration. This allows customers to create minimal solutions that are optimally adapted to the particular task and hardware platform. Furthermore, customers may be interested in adding their own low level components without OS modification. In this article, we present an approach to building modular embedded solutions from heterogeneous components based on the RTOS JetOS. The mechanism of components binding developed by us allows uniting heterogeneous components from different manufacturers within the same section of the address space. This mechanism allows component developer to independently develop their components. And system integrator can independently from developers configure what component he likes to see in OS image and how components should interact.

Keywords: embedded systems, components, RTOS.

DOI: 10.15514/ISPRAS-2017-29(4)-19

For citation: Mallachiev K.A., Pakulin N.V., Khoroshilov A.V., Buzdalov D.V. Using modularization in embedded OS. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 283-294. DOI: 10.15514/ISPRAS-2017-29(4)-19

1. Introduction

Embedded operating systems are built to provide specific functionality on specific hardware. Development of a new OS from scratch for every task and hardware is unwise and operating systems are designed to support several CPU architectures and a lot of peripheral devices in a single distribution. Therefore, OS distribution contains many drivers to support a large number of different hardware. Most of the drivers are not needed for correct OS execution on a specific board. Moreover, many embedded systems are aimed to run in restricted environment, for example with limited memory.

Static OS configuration is used in cases when it is known in advance, on which hardware the OS image is going to be executed. OS configuration is commonly performed by the system integrator. They choose OS features suitable for OS task and drivers for hardware. Only chosen parts will get into final OS image. System integrator doesn’t change OS source code. Static configuration allows keeping final image small.

Safety-critical systems must be certified. For airborne systems there is a standard for certification called DO-178C [1], where OS kernel must be certified by highest level of reliability. Certification is complex and lengthy process. Small change in one part of system leads to recertification of the whole system.

We develop an open-source real-time operating system for civil aircraft airborne computers called JetOS. JetOS is ARINC-653 [2] compliant, supports static configuration and aimed to DO-178 certification.

ARINC-653 specifies interfaces that RTOS (real-time operating system) should provide to avionics software, also the standard specifies some design constrains to the OS. The most pertinent constraint is that application code is executed inside partitions that are isolated from each other by resources and in time.

To simplify and minimize OS kernel and therefore to simplify OS certification process we moved drivers and some services from kernel to special ARINC-653 partitions, called system partitions [3]. Besides drivers system partition contains services such as network stacks, file systems, logging, etc.

System partitions should be certified as well as the kernel. Certification for highly-critical software requires absence of unreachable code. Usage of static configuration of the system partition allows to static selection of required drivers and services, and therefore getting rid of unused code.

It is common that there are many vendors involved in building a specific embedded solution: OS vendor, BSP vendor, device driver developers, system integrator, etc. When services or drivers they are developing are strongly coupled, developers have to interact a lot.

Therefore splitting system partition to independent isolated components seems to be suitable solution. Each driver and service will be in dedicated component. Each component would have a single developer.

Component should interact with each other. Appearance of fixed interface between components would make component development easer. Moreover fixed interface can make system flexible. Statically configured component-based system (in our case system partition) can be flexible in several aspects:

- When there are several components implementing the same interface (e.g. several file systems) and system integrator can choose which component will get into final image.
- When there are several components implementing the same interface, and they all can get into final image. System integrator configure on static, which components interact. For example, if there are two file systems, some component would work with one file system and others with the second one.
- When system integrator can add new component between two interacting, if the new component has a suitable interface. This is useful and can be used, for example, to insert traffic analyzer between protocol stack and network card driver.

Another use-case is to reuse a device driver in an applications stack, such as network card driver in the network stack. Isolated into component the same driver code might serve multiple device instances due to different sets of internal states and configuration parameters. All copies of the component share same driver code, so that each component copy would work with assigned device, would make system scalable and flexible.

Certification of system includes, among others, unit and integration tests. Splitting system partition to components makes certification easier. Component-level tests can be run by component developer. And system integrator doesn’t need to rerun unit tests, he only needs to run integration tests.e.

2. Related Works

Classical distributed components models like Enterprise JavaBeans, CORBA, Corba Component Model and DCOM [4,5,6] define components and interfaces between them. Models allow substituting one component with the other one with the same interfaces. Components configuration dynamically configured by brokers. This approach is not suitable for embedded systems with static configuration.

Ideas to separate OS appeared long ago in microkernels. Microkernel architecture’s [7,8,9] primary goal is to separates OS into independent servers that could be isolated from each other. Servers interact through inter-process communication (IPC). IPC calls are typed and servers with the same interface can substitute one another. But there cannot be two servers with the same interface; therefore this model is not suitable for our tasks too.

VxWorks is a popular embedded operating system. VxWorks board support package (BSP) is divided into components. Components interface is declared in component description language (CDL). BSP developer can construct BSP from existing component and can add their own components. But this system is not flexible; for example, each component has hardcoded in it a list of names of components it interact with, therefore one component cannot be easily substituted in a configuration with another one with the same interfaces.

We are not aware of any component based model with the following set of features:

- Static configuration,
- Low overhead,
- Flexible configuration (in all aspects from introduction),
- Low mishit probability, when component interact with component it not designed to (runtime addressing checks)

3. Basic Capabilities of Component-Based Model

Our model aimed to have small overhead, so it can be suitable for RTOS. In its raw form, our model assumes that there is a lot of similar code written by component developers in C language. To reduce the amount of hand work we generate helper code, based on configuration files. Language, which is used to write configuration files, can be any declarative language; we use YAML for these purposes.

3.1 Component developer view

Model defines component types and component instances. Each component has a unique component type and assigned implementation and any number of instances. Component type is similar to term “class” from object oriented languages and component instance is similar to “class objects”. Component instances share code, but sharing does not apply to some private data, called instance state.

Components interact. The ability of one component to use services of the others is achieved through typed ports. There are two kinds of component ports:

- Input ports, which show that the component provides some functionality. Input ports have assigned handlers implemented by the component, which will be called when some other component calls the interface of the component.
- Output ports, which are used by a component when invokes behavior of another component. The component calls others indirectly, through output ports.

Ports are typed, input port of one component and output of the other one can be connected only if they have the same port type. Port type is called interface. Interface is the set of functions, which input port provides or output port require. Since interface can have several functions, then output port that implements this interface has several assigned handlers, one for each function in interface.

Interface declares as the set of triple of function names, signature, and return types. Example of simple interface declaration can be seen at fig.1.

```
- name: data_sender
  functions:
    - name: send
      return_type: ret_t
      args_type: [int]
```

Fig 1. Data_sender interface with one function ret_t send(int)

Component type declaration contains component name, component instance state structure, and component ports. Output ports are declared as pair of port name and port interface. Input ports are declared as triple (n, I, m): port name n, port interface I, and m a list of pairs of interface function and assigned implementation specified by components function name.

You can see example of component type configuration at fig. 2.

```
name: Filter
state_struct:
  edge: int

input_ports:
  - name: in
    type: data_sender
    implementation:
     send: filter_send

output_ports:
  - name: out
    type: data_sender
```

Fig 2. Component type Filter. Component state contains one field edge. Componet type has single input port called in, port interface is data_sender, fucntion send of data_sender interface is implemented by filter_send function.

During system build configuration files are parsed and corresponding C code is generated:

- C-structure describing component, with name identical to component name. (e.g. structure `Filter` for component Filter)
- Declaration of functions specified in input ports (e.g. declaration of function `filter_send` for component Filter). This declaration enforces naming convention.
- Special function for calling output ports.

Component developers should use only ports to communicate with other components. Direct call of another component might work but is not guaranteed. The component developer is guaranteed only the interfaces. The developer chooses names for ports. Input ports are an entry point to component. Component developer does not use names on input ports. Output ports are used when component should use service of another component. To call the output port a developer should specify output port name, output port function name, and function arguments. Developer should not assume what real function of which component will be called. You can see an example of calling function from output port at fig. 3

```
ret_t filter_send(Filter *self, int data)
{
  ...
  res = Filter_call_out_send(self, data);
  ...
}
```

Fig 3. Call of function send of port out.

3.2 System integrator view

System integrator decides how many instances of each component should be created, and how they are connected. For each component, they choose unique name, and how to initialize its state. System integrator uses instance names and names of their ports to link ports of different instances. All of this information integrator specifies in configuration file. Graphical view of example configuration use can see at fig. 4.

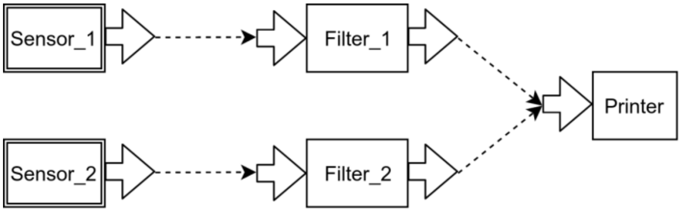


Fig 4. Example linkage configuration. Sensor_1 and Sensor_2 are instances of Sensor component type. Filter_1 and Filter_2 are instance of Filter component type. Sensor_1 ouput port connect to Filter_1 input port. Filter_1 input port connected to Printer. Same for Sensor_2 and Filter_2

4. Advanced capabilities of component based model

4.1 Init function

Instances can have init function: component developers should declare init function name in configuration. At system partition start all init functions of all instances are called sequentially. There is no way to specify dependencies on init (i.e. init of open component should be called before init of the other one) because we assume that components are independent and should not have any dependency.

4.2 Active and reactive components

All components with input ports are *reactive*, i.e. get control by call from other component. Some components are *active*, i.e. the component gets control from OS by some regularities (periodically or by event). Component can be active and reactive at the same time.

There are two types of active components in our model:

- Components which have a special entry point – activity. This type of active components is useful when component instances should do some simple work from time to time (for example, checking whether there are any new networks packets). Component developer declares activity name in configuration. All activities are called sequentially. This type of active components has a big disadvantage: if some instance will freeze in its activity then all instances of this type in the system are going to freeze, so component developer should not use any wait objects in activity.
- Components, which instances create their own threads inside init function. In this case freezing of the instance, which is running in the dedicated thread, will not cause freezing other instances.

4.3 Array of ports

Sometimes component developers need to create configurable number of ports of the same type. We support array of ports, but only for output ports. For calling function of output port array developers should specify index in the array besides port name, function name and function arguments.

Arrays of ports are useful in components like router (at the fig. 5). Router sends data to configurable of instances. Integrator in the configuration specifies number of elements in port array and their linkage with instances.

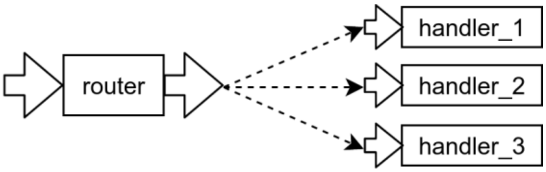


Fig. 5. Router has an array of out port which are connected to instances handler_1, handler_2 and handler_3

4.4 Memory blocks

Component instances in our system cannot use system heap, because there can be heap underflow with many instances and not enough heap size.

Access to heap and physical (for drivers) memory is done through ARINC-653 memory blocks. For each memory block component developer specifies:

- memory block name suffix

- memory size
- memory alignment
- flag, that shows if this memory block used by single instance or shared between instances.
- physical address for drivers working with memory mapped devices. Memory blocks with fixed physical address must be shared.

Name of shared memory blocks is identical to name suffix from configuration. Name of non-shared memory block is concatenation of instance name and memory block name suffix. Instances can access memory blocks by ARINC-653 API specifying memory block name.

4.5 Memory ownership

This part of the paper does not describe a feature of our approach. Here is some consideration on memory ownership.

Let us consider a component based system partition, which implement networking. There can be a track of components: Message_sender →UDP_IP_sender → Eth_sender→ Network_card_driver. Message sender sends pointer message to UDP_IP_sender; UDP_IP_sender prepends message with UPD and IP header and sends message to Eth_sender; Eth_sender prepends message with Ethernet header and sends to Network_card_driver. Should be specified how own memory and responsible for memory allocations.

If UDP_IP_sender and Eth_sender components would allocate buffers in their own memory, then this would greatly complicate their code, as they should also free buffers. Our real time C library does not support memory freeing because memory freeing can make indeterminate amount of time.

To simplify implementation and reduce overhead we used an approach when Message_sender allocates enough memory for all headers (component gets this value from configuration), copies message at the needed offset and pass to next layer pointer to message, message size, prepend and append values. Prepend describes how many bytes before message are allocated. Append describes how many bytes after message are allocated.

UDP_IP_sender to add header moves pointer it gets from Message_sender and decreases prepend value to header size.

5. Future work

We are going to work on supporting component distribution by binary images. This can be used to protect intellectual property of component developer, who does not want to share component source code.

Currently system integrator should specify component instances and their linkage in YAML language. We are going to support AADL language, which allows system integrator to graphically create and link instances. To work with AADL we are going to use MASIW framework. MASIW [10, 11] (MASIW – Modular Avionics System Integrator Workplace) is s an open source Eclipse-based IDE for development and analysis of AADL models.

In addition, we are going to research possibility of using dataflow language to specify component, so that there will be no need to write component implementation in C language

6. Conclusion

In the paper, we presented a component-based approach that was created for JetOS, but can be used in other systems. The approach turned out to be efficient; it has low overhead and make system flexible and scalable while statically configured.

References

[1]. DO-178C, Software Considerations in Airborne Systems and Equipment Certification, January 5, 2012

[2]. Avionics application software standard interface part 1 – required services, ARINC specification 653P1-3, November 15, 2010

[3]. Mallachiev K.M., Pakulin N.V., Khoroshilov A.V. Design and architecture of real-time operating system. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 2, 2016, pp. 181- 192. DOI: 10.15514/ISPRAS-2016-28(2)-12

[4]. J. Siegel, Corba 3 fundamentals and programming, John Wiley & Sons, 2000

[5]. Nanbor Wang, Douglas C. Schmidt, and Carlos O'Ryan. 2001. Overview of the CORBA component model. In Component-based software engineering, George T. Heineman and William T. Councill (Eds.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA 557-571.

[6]. Distributed Component Object Model (DCOM) Remote Protocol Specification (online)

[7]. Alain Gefflaut, Trent Jaeger, Yoonho Park, Jochen Liedtke, Kevin J. Elphinstone, Volkmar Uhlig, Jonathon E. Tidswell, Luke Deller, and Lars Reuther. 2000. The SawMill multiserver approach. In Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system (EW 9). ACM, New York, NY, USA, 109-114. DOI=http://dx.doi.org/10.1145/566726.566751

[8]. J. Liedtke. 1995. On micro-kernel construction. In Proceedings of the fifteenth ACM symposium on Operating systems principles (SOSP '95), Michael B. Jones (Ed.). ACM, New York, NY, USA, 237-250. DOI: http://dx.doi.org/10.1145/224056.224075

[9]. Boule I, Gien M, Guillemont M. CHORUS Distributed Operating Systems, Computing Systems, Vol. I No. 4 Fall 1988

[10]. D. V. Buzdalov, S. V. Zelenov, E. V. Kornykhin, A. K. Petrenko, A. V. Strakh, A. A. Ugnenko, and A. V. Khoroshilov. Tools for system design of integrated modular avionics. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 1, 2014, pp. 201–230 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-6

[11]. Alexey Khoroshilov, Dmitry Albitskiy, Igor Koverninskiy, Mikhail Olshanskiy, Alexander Petrenko, and Alexander Ugnenko. AADL-based toolset for IMA system design and integration. SAE Int. J. Aerosp., 5:294–299, 10 2012.

Использование модульного подхода во встраиваемых операционных системах

[1,2] К.А. Маллачиев <mallachiev@ispras.ru>
[1,2,3] Н.В. Пакулин <npak@ispras.ru>
[1,2,3,4] А.В. Хорошилов <khoroshilov@ispras.ru>
[1] Д.В. Буздалов <buzdalov@ispras.ru>
[1] Институт системного программирования 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.
[3] Московский физико-технический институт,
141701, Россия, Московская обл., г. Долгопрудный, Институтский пер., д. 9.
[4] Национальный исследовательский университет «Высшая школа экономики»
101000, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. Современные операционные системы для встроенных систем могут использоваться для решения задач управления в различных аппаратных контекстах. Управляющие ЭВМ могут различаться архитектурой центрального процессора, составом каналов связи, поддерживаемыми протоколами связи и т. д. Обычно встраиваемые ОС конфигурируются на этапе сборки, позволяя создать образ ОС, предназначенный для выполнения на определенной аппаратной платформе. Эту конфигурацию осуществляет команда, называемая группой системной интеграции. Зачастую ОС для встроенных систем разрабатываются множеством компаний. Если ОС не является модульной, то совместные проектирование, разработка и конфигурирование ОС представляют собой очень сложным задачи. Поддержка модульности и компонентой сборки значительно уменьшает необходимость во взаимодействии между компаниями-разработчиками. Клиентам это позволяет создавать минимальные решения, оптимально адаптированные под особенности задачи и аппаратной платформы. Кроме того, различные производители систем могут быть заинтересованы в том, чтобы внедрять в решение свои специализированные компоненты, в том числе и в бинарном виде, защищающем интеллектуальную собственность разработчиков. В данной статье мы представляем подход к построению модульных решений из гетерогенных компонентов на базе ОС РВ JetOS. Разработанный нами механизм связывания компонентов позволяет объединять гетерогенные компоненты от различных производителей в рамках одного раздела адресного пространства. Этот механизм позволяет разработчикам компонентов осуществлять независимую разработку. А системному интегратору позволяет независимо от разработчиков конфигурировать ОС, выбирая какие компоненты попадут в конечный образ ОС, и как эти компоненты будут взаимодействовать.

Ключевые слова: выстраиваемые системы, модульность, компоненты, ОСРВ

DOI: 10.15514/ISPRAS-2017-29(4)-19

Для цитирования: Маллачиев К.А., Пакулин Н.В., Хорошилов А.В., Буздалов Д.В. Использование модульного подхода во встраиваемых операционных системах. Труды ИСП РАН, том 29, вып. 4, 2017 г., стр. 283-294 (на английском). DOI: 10.15514/ISPRAS-2017-29(4)-19

Список литературы

[1]. DO-178C, Software Considerations in Airborne Systems and Equipment Certification, January 5, 2012
[2]. Avionics application software standard interface part 1 – required services, ARINC specification 653P1-3, November 15, 2010
[3]. Mallachiev K.M., Pakulin N.V., Khoroshilov A.V. Design and architecture of real-time operating system. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 2, 2016, pp. 181- 192. DOI: 10.15514/ISPRAS-2016-28(2)-12
[4]. J. Siegel, Corba 3 fundamentals and programming, John Wiley & Sons, 2000
[5]. Nanbor Wang, Douglas C. Schmidt, and Carlos O'Ryan. 2001. Overview of the CORBA component model. In Component-based software engineering, George T. Heineman and William T. Councill (Eds.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA 557-571.
[6]. Distributed Component Object Model (DCOM) Remote Protocol Specification (online)
[7]. Alain Gefflaut, Trent Jaeger, Yoonho Park, Jochen Liedtke, Kevin J. Elphinstone, Volkmar Uhlig, Jonathon E. Tidswell, Luke Deller, and Lars Reuther. 2000. The SawMill multiserver approach. In Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system (EW 9). ACM, New York, NY, USA, 109-114. DOI= 10.1145/566726.566751
[8]. J. Liedtke. 1995. On micro-kernel construction. In Proceedings of the fifteenth ACM symposium on Operating systems principles (SOSP '95), Michael B. Jones (Ed.). ACM, New York, NY, USA, 237-250. DOI: 10.1145/224056.224075
[9]. Boule I, Gien M, Guillemont M. CHORUS Distributed Operating Systems, Computing Systems, Vol. I No. 4 Fall 1988
[10]. Д.В. Буздалов, С.В. Зеленов, Е.В. Корныхин, А.К. Петренко, А.В. Страх, А.А. Угненко, А.В. Хорошилов. Инструментальные средства проектирования систем интегрированной модульной авионики. Труды ИСП РАН, том 26, вып. 1, 2014 г., стр. 201-230. DOI: 10.15514/ISPRAS-2014-26(1)-6
[11]. Alexey Khoroshilov, Dmitry Albitskiy, Igor Koverninskiy, Mikhail Olshanskiy, Alexander Petrenko, and Alexander Ugnenko. AADL-based toolset for IMA system design and integration. SAE Int. J. Aerosp., 5:294–299, 10 2012.

Debugger for Real-Time OS: Challenges of Multiplatform Support

[1,3] *A.N. Emelenko <emelenko@ispras.ru>*
[1,2] *K.A. Mallachiev <mallachiev@ispras.ru>*
[1,2,3] *N.V. Pakulin <npak@ispras.ru>*
[1] *Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*
[3] *Moscow Institute of Physics and Technology (State University), 9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russia*

Abstract. In this paper, we present our work in developing a debugger for multiplatform real-time operating system Jet OS designed for civil airborne avionics. This system is being developed in the Institute for System Programming of the Russian Academy of Sciences, and it is designed to work within Integrated Modular Avionics (IMA) architecture and implement ARINC-653 API specification. Jet OS supports work on different architectures such as PowerPC, MIPS, x86 and ARM. Debugger for a real-time OS is an important tool in software development process, but debugger for RTOS is more than typical debugger used by desktop developers and we must take into account all specific features of such debugger. Moreover, we must support debugging on many platforms. However, debugger's code has to be developed for each platform and we faced the problem of porting our debugger to different architecture without developing it from scratch. In addition, the debugger must support work within emulators, because it can expand developers' capabilities and increase their efficiency. In this paper, we present the architecture of the debugger for JetOS real-time operating system, which provides capabilities for porting our debugger to a new platform in little to no time, and discuss the challenges imposed by multiplatform support in the OS.

Keywords: debugger; operating systems; multiplatform

DOI: 10.15514/ISPRAS-2017-29(4)-20

For citation: Emelenko A.N., Mallachiev K.A., Pakulin N.V. Debugger for Real-Time OS: Challenges of Multiplatform Support. *Trudy ISP RAN/Proc. ISP RAS,* vol. 29, issue 4, 2017, pp. 295-302. DOI: 10.15514/ISPRAS-2017-29(4)-20

1. Introduction

Application debugger is an indispensable tool in developer's hands. But debugger in a real-time operating system is more than just plain debugger. In this paper we present an on-going project on debugger development for JetOS, a real-time multiplatform operating system that is being developed in the Institute for System Programming of the Russian Academy of Sciences.

JetOS is a prototype operating system for civil airborne avionics. It supports PowerPC, MIPS and x86 platforms. Also, it is designed to work within Integrated Modular Avionics (IMA) architecture and implements ARINC-653 API specification, the de-facto architecture for applied (functional) software.

The primary objectives of ARINC 653 are deterministic behavior and reliable execution of the functional software. To achieve this ARINC-653 imposes strict requirements on time and space partitioning. For instance, all memory allocations and execution schedules are pre-defined statically.

The unit of partitioning in ARINC-653 is called partition. Every partition has its own memory space and is executed in user mode. Partitions consist of one or more processes, operating concurrently, that share the same address space. Processes have data and stack areas and they resemble well-known concept of threads.

Embedded applications might be run in two different environments: in an emulator and on the target hardware. In our project, we use QEMU system emulator. Although QEMU has its own debugger support, its functionality proved to be insufficient for debugging embedded applications. Therefore, we implemented a debugger not only for the target hardware, but for the emulator as well.

2. Specific Features of Debugger for RTOS

Developing a debugger for a real-time OS is not a simple task. During developing, we faced many features of the debugger for RTOS compared to typical debuggers used by desktop developers.

Firstly, there are many interacting processes, which need to be debugged simultaneously. Therefore, our debugger must support capability to switch between them. Moreover, it needs to support work with overlapping virtual addresses space.

Secondly, it is impossible to run the debugger on the same device, where the system runs, because of the lack of on-board resources and lack of interactive facilities. That's why the debugger must be remote.

Thirdly, we must support debugging not only for application developers but also for software developers, such as drivers or kernel developers. As a consequence, the debugger can work with a highly privileged kernel and low-privilege application code.

In addition, the debugger must support work within emulators, because it can expand developers' capabilities and increase their efficiency.

Moreover, JetOS can be run on different processors, because it supports PowerPC, x86 and MIPS architecture. Consequently, the debugger has to run on these platforms too. Thus, the debugger must consider all features of all platforms and emulators, and provide full functionality and correct execution of each process.

In order to meet this requirement, we choose GDB (GNU debugger) for the client part of the debugger, which communicates with the client part of the debugger over a serial port.

3. Related Works

We are not the first to consider the problem of debugging multiplatform RTOS. There are many types of debuggers: some of them don’t have code inserted in the system, such as CodeWarrior, others use remote debugging, for example, the debugger for Pistachio microkernel; besides, there is RTOS debugger for VxWorks.

Here we briefly consider some debuggers for embedded OSes and their primary features.

3.1 CodeWarrior

CodeWarrior [3] is an IDE (integrated development environment) published by Freescale Semiconductor. It is designed to edit, compile and debug software for several microcontrollers and microprocessors (Freescale ColdFire, ColdFire+, Kinetis, Qorivva, PX, Freescale RS08, Freescale S08, and S12Z) and digital signal controllers (DSC MC56F80X and MC5680XX) used in embedded systems. It uses JTAG or BDM interface to control the target system.

CodeWarrior enables the user to debug real-time embedded applications, as well as manipulate the source code to display and change the contents of variables, arrays, and data structures. The developer can also use the debugger to work at the hardware level if necessary.

Via CodeWarrior user can:

- View and change memory, registers and variables.
- Set watchpoints.
- Set breakpoints and conditional breakpoints.
- Break on exceptions.
- Track variables

3.2 VxWorks

VxWorks [5] is a real-time operating system (RTOS) developed as proprietary software by Wind River of Alameda, California, US. It supports Intel (x86, including the new Intel Quark SoC and x86-64), MIPS, PowerPC, SH-4, and ARM architectures. Also, WxWorks includes Wind River Probe JTAG debugger, which supports the latest 32-bit and 64-bit processors based on leading architectures, such as PowerPC, ARM, Intel, and MIPS.

Wind River Probe JTAG debugger is a tool for debug application on bare metall. Developers use JTAG for target hardware communication and USB to connect to their laptop. Probe provides capabilities to control hardware and software in a compact USB JTAG emulator.

Probe debugger implements the following set of features:

- Set hardware and software breakpoints
- Run diagnostic scripts
- Single step through code with a correlated source view
- View and modify CPU core and peripheral registers
- View and modify RAM, cache, and non-volatile memory; supports MMU

3.3 L4Ka::Pistachio

L4Ka::Pistachio [4] is the latest L4 microkernel developed by the System Architecture Group at the University of Karlsruhe. It is the first available kernel implementation of the L4 Version 4 kernel API, which provides support for both 32-bit and 64-bit architectures, multiprocessoring, and superfast local IPC. The current release supports x86-x64 (AMD64/ EM64T, K9 / P4 and higher), x86-x32 (IA32, Pentium and higher), PowerPC 32bit (IBM 440, AMCC Ebony / Blue Gene P).

Pistachio kernel uses kdbg debugger. The debugger directs its I/O via the serial line or the keyboard/screen. It is a local debugger and does not support remote debugging mode, therefore it has a very limited amount of functions.

Debugger for Pistachio can:

- Set breakpoints
- Single step
- Dump memory
- Read registers

Debugger for L4Ka::Pistachio supports two platforms, x86 and PowerPC. It is realized by dividing debugger's code into platform specific and independent parts.

Architecture dependent part of the debugger includes:

- Registers printing
- Single step support
- Memory writing
- TLB printing
- Breakpoints setting

4. Debugger's Challenges for Multiplatform Support

Our debugger consists of two parts – server and client. We use GDB for the client part of our debugger and it has the biggest part of architecture independent code.

In general, messaging mechanism between client and server doesn't change – user communicates with the client, the client sends a special-type packet to the server and waits for the server's answer. The server receives this message, checks control sum, which was sent in this packet, and if it matches the message contents, informs the client that the message was accepted for processing. Then the server performs the action described in the packet and sends its own packet to the client. Understanding of what the client wants from the server is a part of architecture independent code in OS because almost all client requests are standardized, but their execution depends on current hardware.

We can divide our server part of the debugger into 2 parts as shown in Fig.1:

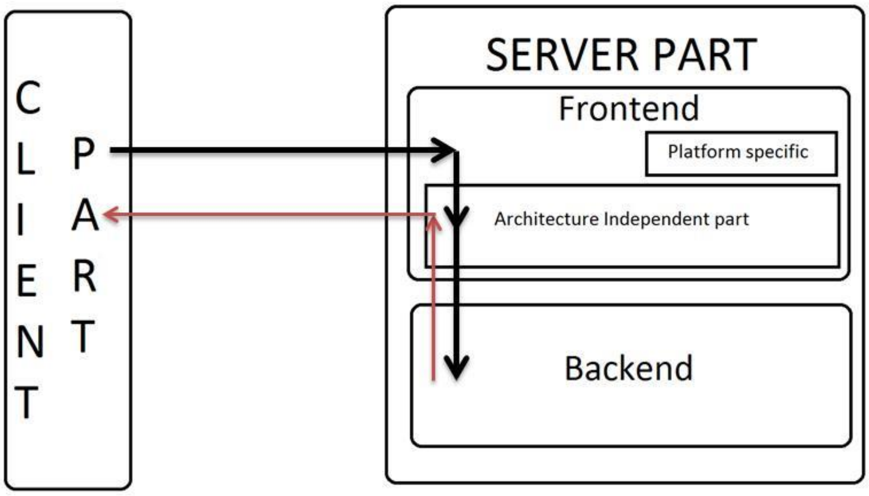


Fig. 1. Debugger's architecture.

4.1 Frontend

This part parses packets, checks control sum, calls the needed function and sends a reply.

Although almost all client-server packets are architecture independent, such requests as registers read/write depend on the target hardware. Therefore, our parser must know not only which architecture is used, but also know the type of packet the client wants to receive, for example, if the client wants to read all registers, the server must send 70 registers on PowerPC and 72 on MIPS.

4.2 Backend

This part of the debugger considers all platform capabilities and uses all available resources.

The largest part of target specific code is responsible for setting breakpoints, watchpoints, single step and read/write in memory. To implement this opportunity not only do we need special code in server part, but we must also change exception handler so that it could distinguish between debugger and regular interrupts.

For example, the single step function is realized in PowerPC architecture via special debug register, used only in this processor. Moreover, breakpoint function needs to set trap instruction where the user wants: for x86 architecture, this is 'int3' instruction, for MIPS – 'break' instruction.

The possibility of debugging applications not only on bare metal but also in emulators, such as QEMU, is also our primary goal. It adds some changes to our realization. For example, there are no debug registers in QEMU for PowerPC architecture, which is used for single step realization on bare metal. Consequently, we need special server part for each realization – on bare metal and QEMU.

5. Debugger's Capabilities

As mentioned above, all debugger's capabilities are available for all supported platforms – x86, PowerPC and MIPS.

5.1 Setting Breakpoints in Partitions and Kernel

Control execution of partitions and kernel is a key feature of debugging. It provides capabilities to more adapted debugger control mechanisms. Moreover, our system supports work with overlapping virtual address spaces, which means that debugger must correctly translate it into physical address.

5.2 Execute the Application Step-by-Step

Run application step by step is a convenient way to control system state and finding bugs. However, next instruction in code might not be the next executable extraction, for example, because of interrupt. Therefore, user can choose to stop on next instruction in code via disabling interrupts or on next executable instruction via platform capabilities.

5.3 Inspect the Application State

In each moment of time, user might want to inspect system state, i.e. memory, registers, stack trace, and threads state.

5.4 Setting Watchpoints

Watchpoints provide a great opportunity to control system state. The developer can use watchpoints to stop execution whenever the value of an expression changes, without having to predict a particular place where this may happen.

6. Debugger's Portability

As already mentioned, JetOS is being developed now, so we don't know the final count of platforms which our system will have to support.

To port our debugger on a new platform, we need only to change the process specific part of the frontend and create the backend part. All other frontend parts are the same for all processes and platforms.

It is easy to imagine the amount of work needed to port the debugger to a new platform with the help of these values:

In PowerPC server part consists of over 2000 lines of code:

- Over 1700 – frontend part
    - 1600 – architecture independent part
    - 100 – platform specific part
- Over 300 – backend part

This separation provides capabilities for porting our debugger to a new platform in little to no time.

7. Conclusion

In this paper, we presented the architecture of remote debugger for JetOS, which included architecture independent code (frontend part) and platform specific code (backend part). This architecture provides capabilities for porting debugger to a new architecture as soon as possible.

One of the next goals is to port our debugger to ARM platform, which support is being developed now.

References

[1]. Lauterbach GmbH, “RTOS debugger for VxWorks”, November 2015 http://www2.lauterbach.com/doc/rtosvxworks.pdf

[2]. Lauterbach GmbH, “RTOS-VxWorks”, 18 August 2014 http://www2.lauterbach.com/pdf/rtos_vxworks.pdf

[3]. Freescale Semiconductor, Inc. CodeWarrior Debugger, December 2, 2004 http://www.nxp.com/assets/documents/data/en/reference-manuals/Engine_PPCRM.pdf

[4]. System Architecture Group University of Karlsruhe. “The L4Ka::Pistachio Microkernel”. May 1, 2003 http://www.l4ka.org/l4ka/pistachio-whitepaper.pdf

[5]. Wind River Systems, Inc “VxWorks Product Overview”, March 2016 http://www.windriver.com/products/product-overviews/VxWorks-Product-Overview-Update.pdf

[6]. Free Software Foundation, Inc. “Debugging with gdb: the gnu Source-Level Debugger”, The Tenth Edition

Отладчик для операционной системы реального времени: проблемы мультиплатформенности

[1,3] *А.Н. Емеленко <emelenko@ispras.ru>*
[1,2] *К.А. Маллачиев <mallachiev@ispras.ru>*
[1,2,3] *Н.В. Пакулин <npak@ispras.ru>*
[1]*Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25.
[2]*Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1.
[3]*Московский физико-технический институт (государственный университет),*
141701, Московская область, г. Долгопрудный, Институтский переулок, д.9.

Аннотация. В этой статье мы расскажем о проекте по разработке отладчика для мультиплатформенной операционной системы реального времени JetOS, созданной для гражданских авиационных систем. Она предназначена для работы в рамках архитектуры Интегрированной Модульной Авионики (ИМА) и реализует ARINC 653 спецификацию API. Эта операционная система разрабатывается в институте системного программирования РАН, и важным шагом в ее создании является разработка инструмента для отладки пользовательских приложений. В этой статье будут рассмотрены проблемы особенностей отладчика для операционной системы реального времени и показаны методы, которыми достигается его мультиплатформенность, а также легкая переносимость на новую платформу. Более того, были рассмотрены другие отладчики для встраиваемых операционных систем, такие как CodeWarrior, отладчики для WxWorks и L4Ka::Pistachio, а также был изучен их функционал. В заключение мы представим наш отладчик, который может работать как в эмуляторе QEMU, используемом для эмуляции окружения для JetOS, так и на целевой машине на всех поддерживаемых платформах. Представленный отладчик является удаленным и построен с использованием структуры GDB, но содержит ряд расширений, специфичных для отладки встроенных приложений. Сама структура отладчика была разделена на архитектурно зависимые и независимые части, что помогает облегчить перенос отладчика на новую платформу. В то же время наш отладчик удовлетворяет большинству требований, налагаемых к отладчику операционной системы реального времени, а также уже используется разработчиками приложений для Jet OS.

Ключевые слова: отладчик; операционные системы; операционная система реального времени; мультиплатформенность.

DOI: 10.15514/ISPRAS-2017-29(4)-20

Для цитирования: Емеленко А.Н., Маллачиев К.А., Пакулин Н.В. Отладчик для операционной системы реального времени: проблемы мультиплатформенности. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 295-302 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-20

Список литературы

[1]. Lauterbach GmbH, “RTOS debugger for VxWorks”, November 2015 http://www2.lauterbach.com/doc/rtosvxworks.pdf

[2]. Lauterbach GmbH, “RTOS-VxWorks”, 18 August 2014 http://www2.lauterbach.com/pdf/rtos_vxworks.pdf

[3]. Freescale Semiconductor, Inc. CodeWarrior Debugger, December 2, 2004 http://www.nxp.com/assets/documents/data/en/reference-manuals/Engine_PPCRM.pdf

[4]. System Architecture Group University of Karlsruhe. “The L4Ka::Pistachio Microkernel”. May 1, 2003 http://www.l4ka.org/l4ka/pistachio-whitepaper.pdf

[5]. Wind River Systems, Inc “VxWorks Product Overview”, March 2016 http://www.windriver.com/products/product-overviews/VxWorks-Product-Overview-Update.pdf

[6]. Free Software Foundation, Inc. “Debugging with gdb: the gnu Source-Level Debugger”, The Tenth Edition

Discovering Near Duplicate Text in Software Documentation*

L.D. Kanteev <lkolt2@mail.ru>
Yu.O. Kostyukov <taxixx@inbox.ru>
D.V. Luciv <d.lutsiv@spbu.ru>
D.V. Koznov <d.koznov@spbu.ru>
M.N. Smirnov <m.n.smirnov@spbu.ru>
Saint Petersburg State University,
7/9 Universitetskaya emb., St. Petersburg, 199034, Russia

Abstract. Development of software documentation often involves copy-pasting, which produces a lot of duplicate text. Such duplicates make it difficult and expensive documentation maintenance, especially in case of long life cycle of software and its documentation. The situation is further complicated by duplicate information frequently being near duplicate, i.e., the same information may be presented many times with different levels of detail, in various contexts, etc. There are a number approaches to deal with duplicates in software documentation. But most of them use software clone detection technique, that is make difficult to provide efficient near duplicate detection: source code algorithms ignore a document structure, and they produce a lot of false positives. In this paper, we present an algorithm aiming to detect near duplicates in software documentation using natural language processing technique called as N-gramm model. The algorithm has a considerable limitation: it only detects single sentences as near duplicates. But it is very simple and may be easily improved in future. It is implemented with use of Natural Language Toolkit (NLTK), and. Evaluation results are presented for five real life documents from various industrial projects. Manual analysis shows 39 % of false positives in automatic detected duplicates. The algorithm demonstrates reasonable performance: documents of 0,8–3 Mb are processed 5–22 min.

Keywords: software documentation, near duplicates, natural language processing, N-gram model.

DOI: 10.15514/ISPRAS-2017-29(4)-21

For citation: Kanteev L.D., Kostyukov Yu.O., Luciv D.V., Koznov D.V., Smirnov M.N. Discovering Near Duplicate Text in Software Documentation. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 303-314. DOI: 10.15514/ISPRAS-2017-29(4)-21

[*] This work is partially supported by RFBR grant No 16-01-00304

1. Introduction

Software projects produce a lot of textual information, and analysis of this data is a truly significant task for practice [1]. One particular problem in this context is software documentation duplicate management. When being developed, a lot of copy-pasted text fragments appeared in software documentation, which is often not tracked properly. According classification from [2], there are different kinds of software documents. For some of them, duplicate text is undesired, while others should contain duplicate text. But in any case duplicates increase documentation complexity and maintenance costs. The situation is further complicated by duplicate information frequently being "near duplicate", i.e., the same information may be presented many times with different levels of detail, in various contexts, etc.

Most popular technique to detect duplicates in software documentation is software clone detection [3]. There are a number of approaches using this technique in software documentation research [4],[5],[6]. However, these approaches operate only with exact duplicates. Near duplicate clone detection techniques [7],[8],[9],[10] are not directly capable of detecting duplicates from text documents as they involve some degree of parsing of the underlying source code for duplicate detection.

In our previous studies [11],[12],[13] we have presented a near duplicate detection approach which is based on software clone detection. We adapted clone detection tool Clone Miner [14] to detect exact duplicates in documents, then near duplicates were extracted as combinations of exact duplicates. However, this approach outcomes a lot of false positives because it can not manage exact duplicate detection and operates with bad-quality "bricks" for combination of near duplicates. Meanwhile false positives' problem is one of the big obstacle of duplicate management in practice [4].

In this paper we suggest an near duplicate detection algorithm based on N-gram model [1]. The algorithm doesn't use software clone detection, omitting the intermediate phases of exact duplicate detection. We have implemented the algorithm using Natural Language Toolkit [15] (NLTK). The algorithm was evaluated on documentation of five industrial projects.

2. Related Work

The problem of duplicate management in software project documents is being actively explored at the moment. Juergens et al. [4] analyze redundancy in requirement specifications. Horie et al. [16] consider the problem of text fragment duplicates in Java API documentation. Wingkvist et al. [5] detect exact duplicates to manage documents maintenance. Rago et al. [17] detect duplicate functionality in textual requirement specifications. However, the problem of near duplicate detection is still open. It is mentioned in [4], and Nosál and Porubän [18] suggest only using near duplicates omitting the way to detect them.

For software engineering, the conceptual background of near duplicate analysis is provided by Bassett [19]. He introduced the terms of archetype (the common part of various occurrences of variable information) and delta (the variation part). Based on this concept, Jarzabek developed an XML-based software reuse method [20]. Koznov and Romanovsky [21],[22] applied the ideas of Bassett and Jarzabek to software documentation reuse, including automated documentation refactoring. However, these studies did not resolve the problem of document duplicate detection.

There are various techniques to detect near duplicate clones in source code. SourcererCC [7] detects near duplicates of code blocks using a static bag-of-tokens strategy that is resilient to minor differences between code blocks. Deckard [8] computes certain characteristic vectors of code to approximate the structure of Abstract Syntax Trees in the Euclidean space. Locality sensitive hashing (LSH) [9] is used to group similar vectors with the Euclidean distance. NICAD [10] is a text-based near duplicate detection tool that also uses a tree-based structural analysis. However, these techniques are not directly capable of detecting duplicates in text documents as they involve some degree of parsing the underlying source code for duplicate detection. A suitable customization for this purpose can be explored in the future.

Finally, there is a need for mature near duplicate detection methods to provide a proper duplicate analysis in software documentation. New information retrieving methods should be applied to increase the search quality. Natural language processing methods appear attractive for that purpose [1].

3. Background

Modern natural language processing and computer linguistics employ numerous standard approaches to analyze and transform texts. One of them is N-gram model [23]. Let us consider the text as a set of sentences. For every sentence the N-gram model includes all sequences (N-grams) consisting of n words, where every next word directly follow to previous one in the same order as in the sentence. Therefore every N-gram is a substring of the correspondent sentence. For example, if we want to detect the fact that two sentence are similar we can to compare their N-gram sets. N-gram model is used to perform different kinds of text analysis.

One of the most common programming tools for practical use of N-gram model is Natural Language Toolkit (NLTK) [15]. It provides a number of standard linguistic operations and is implemented in Python, that makes it easy to integrate NLTK into our Documentation Refactoring Toolkit [24] environment.

4. The Algorithm

The proposed algorithm requires the raw input document to be preprocessed: it should be divided into sentences, the sentences should be divided into words (tokens), and for every sentence an N-gram set is build. The algorithm collects document sentences into groups, if they are close to each other and were likely derived from one source by copy and paste.

The algorithm works as follows. First, it extracts sentences and builds 3-gram set for each of them. After that, for each sentence, the algorithm scans existing groups and chooses the best one, which already contains the largest number of the sentence’s 3-

grams. Then, if the best group already contains at least a half of the sentence’s 3-grams, the sentence is added to this group, and the group's 3-gram set is complemented with the new sentence's 3-grams. When no such group is found, a new group is introduced. Finally, the algorithm outputs the groups that contain two or more sentences. These groups are near duplicate groups.

```
for i = 1 to size(sent) do
    curSent ← sent[i]
    bestOverlap ← 0
    bestGroup ← NULL
    for j = 1 to size(groups) do
        curGroup ← groups[j]
        curIntersect ← intersect(curSent.nGrams, curGroup.nGrams)
        curOverlap ← size(curIntersect) / size(curSent.nGrams)
        if curOverlap > bestOverlap then
            bestOverlap ← curOverlap
            bestGroup ← curGroup
        end if
    end for
    if bestOverlap < 0.5 then
        create new group newGroup
        newGroup.nGrams += curSent.nGrams
        newGroup.sent += curSent
    else
        bestGroup.nGrams += curSent.nGrams
        bestGroup.sent += curSent
    end if
end for
for all G in groups such that size(G) ← 1
    groups −= G
end for
return groups
```

27: Algorithm 1. Specification of the algorithm

Let’s describe the algorithm in more detail. The formal specification of the algorithm is presented on the listing. Below the main functions of the algorithm are briefly considered.

- *intersect(A, B)* function returns elements, which exist in both *A* and *B* sets
- *size(A)* function returns number of elements in the set A
- *sent* is an array of sentences in document text
    - *sent[i].nGrams* is 3-gram set of the *i*-th sentence
- *groups* is an array of near duplicate groups
    - *groups[i].nGrams* is a 3-gram set of *i*-th group

- *groups[i].sent* is a set of sentences of *i*-th group

Details of proposed algorithm are described below:

1. **Lines 1–22:** the main algorithm cycle, which iterates over all sentences of the document.
2. **Lines 5–13:** the cycle for the best group selection. For each groups:
   - 2.1. **Line 7:** intersection of 3-gram set with the 3-gram set of current sentence is calculated.
   - 2.2. **Line 8:** we calculate the ratio of this intersection size to total sentence 3-grams set size.
   - 2.3. **Lines 9–12:** if the current group is the best of processed ones, we remember it.
3. **Line 14:** we check if above ratio is less than 0.5, and:
   - 3.1. **Lines 15–17:** when it is less than 0.5, we create new group and put sentence into it.
   - 3.2. **Lines 19, 20:** otherwise, we put the sentence into the best group found.
4. **Lines 23–25:** groups with single sentence are not near duplicate groups, therefore we remove them.

5. Evaluation

We follow to the GQM framework [25] to organize evaluation of our algorithm. We formulate a set of evaluation questions:

Question 1: How many false positives (incorrect and irrelevant duplicate groups) and meaningful near duplicates are found?

Question 2: What is the performance of the algorithm?

We use the notion *reuse amount* [26] that means the relation of the reusable part to document length. For exact duplicates the reusable part is the total number of symbols, covered by duplicates, for near duplicates we consider only their archetypes. In [4] the same metric is named *clone coverage*.

Following [12], [13] we selected documentation of the four open sources as evaluation objects, but add one more commercial project documentation:

- Linux Kernel documentation (LKD), 892 KB in total [27];
- Zend Framework documentation (Zend), 2924 KB in total [28];
- DocBook 4 Definitive Guide (DocBook), 686 KB in total [29];
- Version Control with Subversion (SVN), 1810 KB in total [30];
- Commercial project user guide (CProj), 164 KB in total.

To answer **question 1**, we performed an manual analysis of near duplicate detected. The results are presented in Table 1.

The table includes column *Document* (evaluation documents) and two sections: *Proposed algorithm* (data concerning algorithm presented in the paper) and *Manual analysis* (results of manual analysis of the algorithm output). The *Proposed algorithm* section is organized as follows:

- *automatically detected* shows numbers of groups, which algorithm found;
- *raw reuse amount* contains reuse amount values for the evaluated documents.

The *Manual analysis* section contains the following columns:

- *markup-only* contains numbers of groups without human-readable text (they only contain markup);
- *irrelevant* presents numbers of false-positive groups, which were detected by human during manual revision of algorithm output;
- *total meaningful* shows number of meaningful duplicates, manually detected analyzing algorithm output;
- *meaningful reuse amount* presents reuse amount values for meaningful near duplicates detected.

Table 1. Near-duplicate groups detected

Document	Proposed algorithm						Manual analysis	
	Automatically detected	Raw reuse amount	Markup-only	Irrelevant	Total meaningful	Meaningful reuse amount	Total meaningful	Meaningful reuse amount
LKD	189	18.9%	20.1%	13.2%	66.7%	7.7%	15	5.1%
Zend	601	14.5%	10.3%	26.5%	63.2%	8.6%	27	2.1%
DocBook	73	13.0%	13.7%	32.9%	53.4%	3.2%	12	1.7%
SVN	349	10.2%	27.8%	21.5%	50.7%	5.0%	16	2.3%
CProj	72	38.3%	0.0%	29.2%	70.8%	29.5%	9	14.1%
Average		19.0%	14.4%	24.6%	61.0%	10.8%		5.0%

14.4% of groups contain no human-readable text, but only markup, 24.6% of groups contain text which is similar, but this is just formal similarity, and duplicates of those groups are not semantically connected. Remaining 61% of groups are meaningful duplicate groups. For documents of different sizes their count varies from few dozens to several hundreds depending on the size and nature of document, therefore we can say that proposed algorithm detects considerable amount of near duplicates, and most of them are meaningful. The reuse amount has been decreased in 2 times after manual processing. These data indicates the false positive problem need to be resolved for the algorithm.

Finally, to answer **question 2** we estimated the working time of the algorithm with the evaluation documents. For our experiments we used the usual work station Intel i5-2400, 3.10GHz, RAM 4 GiB, Windows 10. Our estimation results are presented in table 2. The first column of the table contains the acronyms of the documents to be evaluated. The second one contains the size of the documents. The third column presents the algorithm processing time values. The forth column presents the processing speed. The processing speed depends on two parameters: the size of the document and the reuse amount. It decreases when the document size grows and as the reuse amount increases. The first statement is obvious. The second one follows from the fact that, roughly speaking, the larger the reuse amount is, the fewer groups of single sentence exist, and therefore number operations in cycle of the best group selection (see listing 1, lines 5-13) decreases. However, this is a rough estimation because the size of the groups also contributes to the processing speed. And we cannot say for certain whether or not a larger reuse amount might compensate for a larger document size. Among the five documents presented in table 2, we can see our assumption confirmed. In the case of these documents, the processing speed decreases as the document size increases, with one exception. The processing speed of the algorithm for Zend was higher than that for SVN, although the size of the Zend document was bigger than that of SVN. At the same time, the reuse amount of Zend is substantially higher than that of SVN. Also the assumption concerning the reuse amount works well in our experiments carried out outside of results presented in this paper. However, further research is needed to verify this assumption. In addition, implementation factors need to be explored, which can influence the algorithm performance. Finally, the performance of the algorithm appears sufficient for practical applications. The algorithm demonstrates an acceptable processing time for rather large documents, i.e. from 1 to 3 Mb. Larger documents are quite rare in practice.

Table 2. Performance analysis

Document	Size, Kb	Processing time, min	Processing speed, Kb/min
LKD	892	5.30	168.35
Zend	2924	22.14	132.08
DocBook	686	2.02	339.60
SVN	1810	17.14	105.59
CProj	164	0.17	946.15

6. Conclusion

We have presented an algorithm for the detection of near duplicates in software documentation based on N-gram model. The proposed algorithm is close to the naive voting clustering algorithm [31], using a similarity measure resembling the Jaccard index [32]. Compared to [12],[13], the algorithm looks much simpler, while also making use of the techniques and apparatus conventionally used for text analysis. It should be noted, the algorithm has a considerable limitation: it only detects single sentences as near duplicates. Our primary goal for future research is to extend the algorithm to make possible processing arbitrary text fragments. Here are some additional future directions of the research:

1. It is necessary to resolve false positives problem. The algorithm output should be compared to manual document analysis.
2. Classification of false positives and meaningful near duplicates should be developed. False positives may include markup, document metadata, etc. Meaningful near duplicates usually describe entities of the same nature (function descriptions, command line parameters, data type specifications, etc.).
3. Improvement of experiment model should be performed. For example, Juergens et al. [4] spend much effort to obtain objective results in analyzing duplicates of real industry documents.

Research results could be applied in various fields of software engineering, e.g. in model based testing [33],[34] to provide correctness of initial requirement specifications, which are used for test generation.

References

[1] Wagner S., Fernández D.M. Analysing Text in Software Projects. Preprint, 2016. URL: https://arxiv.org/abs/1612.00164

[2] Parnas D. L. Precise Documentation: The Key To Better Software. Nanz S. (ed.) The Future of Software Engineering, Springer, 2011. DOI: 10.1007/978-3-642-15187-3_8

[3] Akhin, M., Itsykson, V. Clone Detection: Why, What and How? Proceedings of CEE-SECR’10, 2010, pp. 36–42. DOI: 10.1109/CEE-SECR.2010.5783148

[4] Juergens E. et al. Can clone detection support quality assessments of requirements specifications? Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering, 2010, vol. 2, pp. 79–88.

[5] Wingkvist A., Ericsson M., Lincke R., Löwe W. A Metrics-Based Approach to Technical Documentation Quality. Proceedings of 7th International Conference on the Quality of Information and Communications Technology, 2010, pp. 476–481.

[6] Nosál M., Porubän J. Preliminary report on empirical study of repeated fragments in internal documentation. Proceedings of the Federated Conference on Computer Science and Information Systems, Gdansk, 2016, pp. 1573–1576.

[7] Sajnani H., Saini V., Svajlenko J., Roy C.K., Lopes C.V. Sourcerercc: Scaling code clone detection to big-code. Proceedings of the 38th International Conference on Software Engineering, ACM, New York, USA, 2016, pp. 1157–1168. DOI: 10.1145/2884781.2884877

[8] Jiang L., Misherghi G., Su Z., Glondu S. DECKARD: Scalable and accurate tree-based detection of code clones. Proceedings of 29th International Conference on Software Engineering. Institute of Electrical and Electronics Engineers, 2007, pp. 96–105. DOI: 10.1109/ICSE.2007.30

[9] Huang T.K., Rahman M.S., Madhyastha H.V., Faloutsos M., Ribeiro B. An analysis of socware cascades in online social networks. Proceedings of the 22Nd International Conference on World Wide Web, 2013, pp. 619–630.

[10] Cordy J.R., Roy C.K.: The NiCad clone detector. Proceedings of the 19th IEEE International Conference on Program Comprehension. Institute of Electrical and Electronics Engineers, 2011, pp. 219–220. DOI: 10.1109/ICPC.2011.26

[11] Lutsiv D.V., Koznov D.V., Basit H.A., Lieh O.E., Smirnov M.N., Romanovsky K.Yu. An approach for clone detection in documentation reuse. Nauchno-tehnicheskij vestnik informacionnyh tehnologij, mehaniki i optiki [Scientific and Technical Journal of Information Technologies, Mechanics and Optics] vol. 92, issue 4, 2014, pp. 106–114 (in Russian).

[12] Koznov D. et al. Clone detection in reuse of software technical documentation. Mazzara M., Voronkov A. (eds.), International Andrei Ershov Memorial Conference on Perspectives of System Informatics, 2015; Lecture Notes in Computer Science, vol. 9609, 2016, pp. 170–185. DOI: 10.1007/978-3-319-41579-6_14

[13] Luciv D., Koznov D., Basit H.A., Terekhov A.N. On fuzzy repetitions detection in documentation reuse. Programming and Computer Software, vol. 42, issue 4, 2016, pp. 216–224. DOI: 10.1134/s0361768816040046

[14] Basit H.A., Smyth W.F., Puglisi S.J., Turpin A., Jarzabek S. Efficient Token Based Clone Detection with Flexible Tokenization. Proceedings of ACM SIGSOFT International Symposium on the Foundations of Software Engineering, ACM Press, 2007, pp. 513–516. DOI: 10.1145/1295014.1295029

[15] Natural Language Toolkit, URL: http://nltk.org/

[16] Horie M., Chiba S. Tool support for crosscutting concerns of API documentation. Proceedings of 9th International Conference on Aspect-Oriented Software Development, 2010, pp. 97–108. DOI: 10.1145/1739230.1739242

[17] Rago A., Marcos C., Diaz-Pace J.A. Identifying duplicate functionality in textual use cases by aligning semantic actions. International Journal on Software and Systems Modeling, vol. 15, issue 2, 2016, pp. 579–603. DOI: 10.1007/s10270-014-0431-3

[18] Nosál' M., Porubän J. Reusable software documentation with phrase annotations. Open Computer Science, vol. 4, issue 4, 2014, pp. 242-258. DOI: 10.2478/s13537-014-0208-3

[19] Bassett P. Framing software reuse – lessons from real world. Prentice Hall, 1996. ISBN: 0-13-327859-X

[20] Jarzabek S., Bassett P., Zhang H., Zhang W. XVCL: XML-based Variant Configuration Language. Proceedings of 25th International Conference on Software Engineering, 2003, pp. 810–811. DOI: 10.1109/ICSE.2003.1201298

[21] Koznov D., Romanovsky K.. DocLine: A Method for Software Product Lines Documentation Development. Programming and Computer Software, vol. 34, issue 4, 2008, pp. 216–224. DOI: 10.1134/S0361768808040051

[22] Romanovsky K., Koznov D., Minchin L. Refactoring the Documentation of Software Product Lines. Central and East European Conference on Software Engineering Techniques, Brno (Czech Republic), 2008; Lecture Notes in Computer Science, vol. 4980, Springer, 2011, pp. 158–170. DOI: 10.1007/978-3-642-22386-0_12

[23] Broder A.Z. et al. Syntactic clustering of the web. Computer Networks and ISDN Systems. vol. 29, issue 8, 1997, pp. 1157–1166. DOI: 10.1016/S0169-7552(97)00031-7

[24] Documentation Refactoring Toolkit,
URL: http://www.math.spbu.ru/user/kromanovsky/docline/index_en.html

[25] Basili V., Caldiera G., Rombach H. The Goal Question Metric Approach. Encyclopedia of Software Engineering, Wiley, 1994. DOI: 10.1002/0471028959.sof142

[26] Frakes W., Terry C.. Software reuse: metrics and models. ACM Computing Surveys, vol. 28, issue 2, 1996, pp. 415–435. DOI: 10.1145/234528.234531

[27] Linux Kernel Documentation, snapshot on Dec 11, 2013.
URL: https://github.com/torvalds/linux/tree/master/Documentation/DocBook/

[28] Zend PHP Framework documentation, snapshot on Apr 24, 2015. URL: https://github.com/zendframework/zf1/tree/master/documentation

[29] DocBook Definitive Guide, snapshot on Apr 24, 2015. URL: http://sourceforge.net/p/docbook/code/HEAD/tree/trunk/defguide/en/

[30] SVN Book, snapshot on Apr 24, 2015. URL: http://sourceforge.net/p/svnbook/source/HEAD/tree/trunk/en/book/

[31] Braun R.K., Kaneshiro R. Exploiting topic pragmatics for new event detection. Technical report. National Institute of Standards and Technology, Topic Detection and Tracking Workshop, 2004.

[32] Jaccard P. Distribution de la flore alpine dans le Bassin des Dranses et dans quelques regions voisines [Distribution of Alpine flora in the Dranses Basin and some neighboring regions]. Bulletin de la Société Vaudoise des Sciences Naturelles [Bulletin of the Vaudois Society of Natural Sciences], vol. 140, issue 37, 1901, pp. 241–272 (in French)

[33] Drobintsev P.D., Kotlyarov V. P., Letichevsky A.A. A formal approach to test scenarios generation based on guides. Automatic Control and Computer Sciences, vol. 48, issue 7, 2014, pp. 415–423. DOI: 10.3103/S0146411614070062

[34] Zelenov S.V., Silakov D.V., Petrenko A.K., Conrad M., Fey I. Automatic test generation for model-based code generators. Proceedings of 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, pp. 75–81. DOI: 10.1109/ISoLA.2006.70

Обнаружение неточно повторяющегося текста в документации программного обеспечения *

Л.Д. Кантеев <lkolt2@mail.ru>
Ю.О. Костюков <taxixx@inbox.ru>
Д.В. Луцив <d.lutsiv@spbu.ru>
Д.В. Кознов <d.koznov@spbu.ru>
М.Н. Смирнов <m.n.smirnov@spbu.ru>
Санкт-Петербургский государственный университет,
199034, Россия, Санкт-Петербург, Университетская набережная 7/9

Аннотация. При создании документации программного обеспечения часто применяется копирование и вставка с последующим редактированием, в результате чего возникает много повторяющегося текста. Такие повторы усложняют и удорожают поддержку документации, особенно в случае длительных жизненных циклов программного обеспечения и документации. Ещё более усложняет ситуацию то, что зачастую информация повторяется приблизительно, т.е. одна и та же информация может быть многократно представлена с разными уровнями детализации, в различных контекстах и т.д. В данной работе предложен алгоритм, предназначенный для обнаружения неточных повторов в документации программного обеспечения. Алгоритм основан на модели N-грамм и реализован с использованием Natural Language Toolkit. Алгоритм апробирован на документации нескольких проектов с открытым исходным кодом.

Ключевые слова: документация программного обеспечения, нечёткие повторы, обработка текстов на естественных языках, модель N-грамм.

[*] Работа частично поддержана грантом РФФИ №16-01-00304

DOI: 10.15514/ISPRAS-2017-29(4)-21

Для цитирования: Кантеев Л.Д., Костюков Ю.О., Луцив Д.В., Кознов Д.В., Смирнов М.Н. Обнаружение неточно повторяющегося текста в документации программного обеспечения. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 303-314 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-20

Список литературы

[1] Wagner S., Fernández D.M. Analysing Text in Software Projects. Preprint, 2016. URL: https://arxiv.org/abs/1612.00164

[2] Parnas D. L. Precise Documentation: The Key To Better Software. Nanz S. (ed.) The Future of Software Engineering, Springer, 2011. DOI: 10.1007/978-3-642-15187-3_8

[3] Akhin, M., Itsykson, V. Clone Detection: Why, What and How? Proceedings of CEE-SECR'10, 2010, pp. 36–42. DOI: 10.1109/CEE-SECR.2010.5783148

[4] Juergens E. et al. Can clone detection support quality assessments of requirements specifications? Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering, 2010, vol. 2, pp. 79–88.

[5] Wingkvist A., Ericsson M., Lincke R., Löwe W. A Metrics-Based Approach to Technical Documentation Quality. Proceedings of 7th International Conference on the Quality of Information and Communications Technology, 2010, pp. 476–481.

[6] Nosál M., Porubän J. Preliminary report on empirical study of repeated fragments in internal documentation. Proceedings of the Federated Conference on Computer Science and Information Systems, Gdansk, 2016, pp. 1573–1576.

[7] Sajnani H., Saini V., Svajlenko J., Roy C.K., Lopes C.V. Sourcerercc: Scaling code clone detection to big-code. Proceedings of the 38th International Conference on Software Engineering, ACM, New York, USA, 2016, pp. 1157–1168. DOI: 10.1145/2884781.2884877

[8] Jiang L., Misherghi G., Su Z., Glondu S. DECKARD: Scalable and accurate tree-based detection of code clones. Proceedings of 29th International Conference on Software Engineering. Institute of Electrical and Electronics Engineers, 2007, pp. 96–105. DOI: 10.1109/ICSE.2007.30

[9] Huang T.K., Rahman M.S., Madhyastha H.V., Faloutsos M., Ribeiro B. An analysis of socware cascades in online social networks. Proceedings of the 22Nd International Conference on World Wide Web, 2013, pp. 619–630.

[10] Cordy J.R., Roy C.K.: The NiCad clone detector. Proceedings of the 19th IEEE International Conference on Program Comprehension. Institute of Electrical and Electronics Engineers, 2011, pp. 219–220. DOI: 10.1109/ICPC.2011.26

[11] Луцив Д.В., Кознов Д.В., Басит Х.А., Ли О.Е., Смирнов М.Н., Романовский К.Ю. Метод поиска повторяющихся фрагментов текста в технической документации. Научно-технический вестник информационных технологий, механики и оптики, т. 92, вып. 4, 2014, стр. 106–114.

[12] Koznov D. et al. Clone detection in reuse of software technical documentation. Mazzara M., Voronkov A. (eds.), International Andrei Ershov Memorial Conference on Perspectives of System Informatics, 2015; Lecture Notes in Computer Science, vol. 9609, 2016, pp. 170–185. DOI: 10.1007/978-3-319-41579-6_14

[13] Луцив Д.В., Кознов Д.В., Басит Х.А., Терехов А.Н. Задача поиска нечётких повторов при организации повторного использования документации. Программирование, т. 42, № 4, 2016, стр. 39–49.

[14] Basit H.A., Smyth W.F., Puglisi S.J., Turpin A., Jarzabek S. Efficient Token Based Clone Detection with Flexible Tokenization. Proceedings of ACM SIGSOFT

International Symposium on the Foundations of Software Engineering, ACM Press, 2007, pp. 513–516. DOI: 10.1145/1295014.1295029

[15] Natural Language Toolkit, URL: http://nltk.org/

[16] Horie M., Chiba S. Tool support for crosscutting concerns of API documentation. Proceedings of 9th International Conference on Aspect-Oriented Software Development, 2010, pp. 97–108. DOI: 10.1145/1739230.1739242

[17] Rago A., Marcos C., Diaz-Pace J.A. Identifying duplicate functionality in textual use cases by aligning semantic actions. International Journal on Software and Systems Modeling, vol. 15, issue 2, 2016, pp. 579–603. DOI: 10.1007/s10270-014-0431-3

[18] Nosál' M., Porubän J. Reusable software documentation with phrase annotations. Open Computer Science, vol. 4, issue 4, 2014, pp. 242-258. DOI: 10.2478/s13537-014-0208-3

[19] Bassett P. Framing software reuse – lessons from real world. Prentice Hall, 1996. ISBN: 0-13-327859-X

[20] Jarzabek S., Bassett P., Zhang H., Zhang W. XVCL: XML-based Variant Configuration Language. Proceedings of 25th International Conference on Software Engineering, 2003, pp. 810–811. DOI: 10.1109/ICSE.2003.1201298

[21] Кознов Д.В., Романовский К.Ю. DocLine: метод разработки документации семейства программных продуктов. Программирование, т. 34, вып. 4, 2008, С. 1–13.

[22] Romanovsky K., Koznov D., Minchin L. Refactoring the Documentation of Software Product Lines. Central and East European Conference on Software Engineering Techniques, Brno (Czech Republic), 2008; Lecture Notes in Computer Science, vol. 4980, Springer, 2011, pp. 158–170. DOI: 10.1007/978-3-642-22386-0_12

[23] Broder A.Z. et al. Syntactic clustering of the web. Computer Networks and ISDN Systems. vol. 29, issue 8, 1997, pp. 1157–1166. DOI: 10.1016/S0169-7552(97)00031-7

[24] Documentation Refactoring Toolkit, URL: http://www.math.spbu.ru/user/kromanovsky/docline/index.html

[25] Basili V., Caldiera G., Rombach H. The Goal Question Metric Approach. Encyclopedia of Software Engineering, Wiley, 1994. DOI: 10.1002/0471028959.sof142

[26] Frakes W., Terry C.. Software reuse: metrics and models. ACM Computing Surveys, vol. 28, issue 2, 1996, pp. 415–435. DOI: 10.1145/234528.234531

[27] Linux Kernel Documentation, snapshot on Dec 11, 2013. URL: https://github.com/torvalds/linux/tree/master/Documentation/DocBook/

[28] Zend PHP Framework documentation, snapshot on Apr 24, 2015. URL: https://github.com/zendframework/zf1/tree/master/documentation

[29] DocBook Definitive Guide, snapshot on Apr 24, 2015. URL: http://sourceforge.net/p/docbook/code/HEAD/tree/trunk/defguide/en/

[30] SVN Book, snapshot on Apr 24, 2015. URL: http://sourceforge.net/p/svnbook/source/HEAD/tree/trunk/en/book/

[31] Braun R.K., Kaneshiro R. Exploiting topic pragmatics for new event detection. Technical report. National Institute of Standards and Technology, Topic Detection and Tracking Workshop, 2004.

[32] Jaccard P. Distribution de la flore alpine dans le Bassin des Dranses et dans quelques regions voisines. Bulletin de la Société Vaudoise des Sciences Naturelles, vol. 140, issue 37, 1901, pp. 241–272 (франц.)

[33] Drobintsev P.D., Kotlyarov V. P., Letichevsky A.A. A formal approach to test scenarios generation based on guides. Automatic Control and Computer Sciences, vol. 48, issue 7, 2014, pp. 415–423. DOI: 10.3103/S0146411614070062

[34] Zelenov S.V., Silakov D.V., Petrenko A.K., Conrad M., Fey I. Automatic test generation for model-based code generators. Proceedings of 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, pp. 75–81. DOI: 10.1109/ISoLA.2006.70

The Program for Public Mood Monitoring through Twitter Content in Russia

S.I. Smetanin <sismetanin@gmail.com>
National Research University Higher School of Economics,
20, Myasnitskaya st., Moscow, 101000 Russia

Abstract. With the popularization of social media, a vast amount of textual content with additional geo-located and time-stamped information is directly generated by human every day. Both tweet meaning and extended message information can be analyzed in a purpose of exploration of public mood variations within a certain time periods. This paper aims at describing the development of the program for public mood monitoring based on sentiment analysis of Twitter content in Russian. Machine learning (naive Bayes classifier) and natural language processing techniques were used for the program implementation. As a result, the client-server program was implemented, where the server-side application collects tweets via Twitter API and analyses tweets using naive Bayes classifier, and the client-side web application visualizes the public mood using Google Charts libraries. The mood visualization consists of the Russian mood geo chart, the mood changes plot through the day, and the mood changes plot through the week. Cloud computing services were used in this program in two cases. Firstly, the program was deployed on Google App Engine, which allows completely abstracts away infrastructure, so the server administration is not required. Secondly, the data is stored in Google Cloud Datastore, that is, the highly-scalable NoSQL document database, which is fully integrated with Google App Engine.

Keywords: sentiment analysis; public mood; mood patterns; twitter; social media

DOI: 10.15514/ISPRAS-2017-29(4)-22

For citation: Smetanin S.I. The Program for Public Mood Monitoring through Twitter Content in Russia. *Trudy ISP RAN/Proc. ISP RAS,* vol. 29, issue 4, 2017, pp. 315-324. DOI: 10.15514/ISPRAS-2017-29(4)-22

1. Introduction

With the popularization of social media, particularly the micro-blogging website Twitter, a vast amount of content is directly generated by people every day. In addition to textual information, which seems to have affective component, Twitter messages are also time-stamped and geo-located. Consequently, both tweets meaning and extended information about a message can be analyzed in a purpose of scientific studies in general and in the exploration of public mood variations particularly.

In data mining, the usage of social media to analyze and predict political events is becoming more popular in recent times. During the Brexit referendum in the United Kingdom, the researchers consider changes to the public mood within the contents of Twitter [13]. They measure the appearance of positive and negative affect in various geographic regions of the United Kingdom, at hourly intervals. According to the results, there are three key times in the period leading up to and including the EU referendum, each of which was characterized by an increase in negative affect with a corresponding loss of positive affect.

The paper [12] describes an empirical study of Relationship between Twitter mood and stock market from an Indian context. Using Twitter as a source of the news, the authors have extracted the polarity of messages and have found a significant correlation with stock market movement measured in the major stock indices of India. In addition, the correlation of the sentiment with other macroeconomic factors like Gas and Oil Price was established.

Academics from the University of Bristol have published two papers with analysis of periodic patterns in daily media content and consumption under the ThinkBIG project [17]. The first paper [6] was focused on the scrutiny of 87 years of the United States and United Kingdom newspapers between 1836 and 1922. Studies have found people's behavior were strongly correlated with the weather and seasons. In the second paper [7], presented at 2016 IEEE International Conference on Data Mining, the authors pay their attention to discovering mental health changes. The team analyzed Twitter content in the United Kingdom and Wikipedia access over four years using data mining and sentiment analysis techniques. They found that negative sentiment tends to be overexpressed in the winter with the peak value in November, while more aggressive emotions like anxiety and anger seem to be overexpressed between September and April. To conclude, both papers states that people's collective behavior follows strong periodic patterns.

This paper describes the development of the program for monitoring peoples' mood through Twitter content in Russian. This paper aims at implementing the software product for exploring the temporal and geographical mood patterns in Russia using machine learning techniques. In contrast with issues mentioned above, this program is designed to process Twitter data in the online mode, i.e. to receive data directly from Twitter API in real time, rather than analyze the pre-collected messages corpus.

The paper is organized as follows. In section 2 the program implementation, methodology, and data collection are described. Section 3 is focused on results and further ways of research. The limitations of this paper are provided in section 4.

2. Implementation, data, and methodology

With the popularization of social media, particularly the micro-blogging website Twitter, a vast amount of content is directly generated by people every day. In addition to textual information, which seems to have affective component, Twitter messages are also time-stamped and geo-located. Consequently, both tweets meaning and extended information about a message can be analyzed in a purpose of scientific studies in general and in the exploration of public mood variations particularly.

The client-server model was implemented for this project, where the server-side application collects and analyzes Twitter content, and the client-side web application visualizes results. Python was selected as a preferred programming language because of its cross-platform operability, open source code and a vast number of third-party libraries. The Google App Engine [9] cloud platform was used to run and host this project on Google's infrastructure in Python runtime environment. The applications data is stored in Google App Engine Cloud Datastore [4], that is, a high-performance database.

Fig. 1 illustrates the process of public mood monitoring; it's clear that it can be divided into several parts. Firstly, messages obtained via Twitter API [19] using Python-based client library. Secondly, the identification of a federal subject for each obtained message is performed. Thirdly, sentiment analysis is executed. Fourthly, the information of emotional polarity of messages is stored in the database. At the last step, the client-side application visualizes results. Details for these parts are given in the following sections.

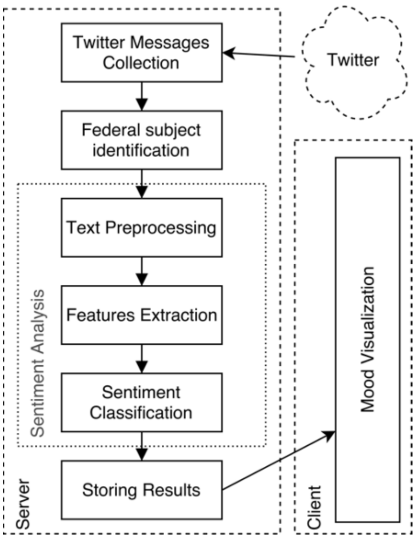


Fig. 1. The program architecture

2.1 Twitter messages collection

Twitting with a location is the geotagging feature in the Twitter platform. On the one hand, this feature helps to provide more meaningful experience for users by making messages more contextual. On the other hand, it makes possible for researchers to analyze Twitter content from the location-based point of view. In order to use the Tweeting with location feature users must opt-in, i.e. turn location "on". The location will be displayed with users Tweets only in case if they give explicit permission for location extraction. Twitter tracks their location via mobile geo-services or IP.

It's common for IT companies to release its API to the public so that other software developers can design products that are powered by its service. To access Twitter content programmatically it's necessary to register the developer application in Twitter Developers Console. Using credentials from the registered application it's possible to interact with Twitter API from the code of the program. The open-sourced library Tweepy [18] was used in this project to communicate with the Twitter platform and use its' API. The cron job, that is, time-based job scheduler in Unix-like computer OS, is searching and collecting new tweets in Russian with geotagging information via Tweepy every minute. In other words, the information about newly published messages is updated in the program every minute.

2.2 Federal subject identification by message coordinates

For each message collected at the previous step the administrative-territorial entity should be defined according to ISO 3166-2:RU standard, that is, part of ISO 3166 standard published by the International Organization for Standardization, which describes the principal subdivisions of all countries coded in ISO 3166-1.

Due to high implementation complexity, it was decided to use existing geographical services to identify federal subjects' codes. The GeoNames [8] worldwide geographical database was selected for identification of the federal subject code by message latitude and longitude values. This service provides developers with HTTP REST API, which includes identification of the country ISO code and the administrative subdivision of any given point. According to GeoNames terms and conditions of use, there are 30000 requests daily limit and 2000 hourly limit for the code identification functional.

2.3 Sentiment Analysis

The sentiment analysis process can be divided into three steps. At the first step, text preprocessing for collected messages is executed to prepare textual information for sentiment analysis. At the second step, classification features are extracted from prepared messages. At the last step, sentiment classification for each message is performed. The detail description of the steps is as follows.

1) Text preprocessing

Texts generated by humans in social media sites contain lots of noise that can significantly affect the results of the sentiment classification process. Moreover, depending on the features generation approach, every new word seems to add at least one new dimensional, that makes the representation of texts is sparse and high-dimensional, consequently, the task of the classifier has become more complex.

According to [10], text preprocessing has been found crucial on the sentiment classification performance.

To prepare messages, such text preprocessing techniques as reverting repeated letters, removing URLs, removing numbers, converting to lowercase, word normalization and stemming were used in this program. Removing and replacing tasks was performed using regular expressions. The morphological analyzer PyMorphy2 [11]was used for words normalization. Stemming of normalized words was performed using NLTK Python library [16].

2) Features extraction

A basic step for a static natural language processing task tends to be the conversion of raw text into features, which provides a machine learning model with a simpler, more comprehensible view of the text. The bag-of-words model was used to calculate texts embedding using unigrams and bigrams.

3) Sentiment classification

In this project, the multinomial Naïve Bayes classification algorithm for binary sentiment analysis task was used because of its tendency to perform significantly well in the texts classification task and wide usage [20], [2], [14]. The basic idea of Naïve Bayes technique is to find the probabilities of classes assigned to texts by using the joint probabilities of words and classes [5]. Consider the given data point *x* and class $c \in C$. The starting point is Bayes' theorem for conditional probability which estimates as follows:

$$P(c|x) = \frac{P(x|c)}{P(x)}$$

$$P(x|c) = \frac{count(x,c)}{count(c)}$$

Where *count(x, c)* is the count of word *x* in class *c; count(c)* is a count of all words in class *c*. For texts with unknown words, the estimation (2) might be problematic because it would give zero probability. The usage of Laplace smoothing is a common way to solve this problem (3).

$$P(x|c) = \frac{count(x,c) + 1}{count(c) + |V| + 1}$$

Where $|V|$ is the length of vocabulary in training set.

From the assumption of word independence, it appears that for data point $x = \{x_1, x_2, ..., x_i\}$ the probability of each of its features to occur in the given class is independent. Thus, the estimation of this probability can be calculated as follows:

$$P(c|x) = P(c)\prod P(x_i|C)$$

In this context, that means the final equation for the class chosen by a naive Bayes classifier is (5).

$$c_{nb} = \underset{c \in C}{\operatorname{argmax}} P(c)\prod P(x_i|c)$$

To avoid underflow and increase speed, the Naive Bayes calculations are performed in the log space (6).

$$c_{nb} = \underset{c \in C}{\operatorname{argmax}} (\log \mathrm{P(c)} + \sum \log P(x_i|c))$$

The Naive Bayes classifier was trained on the corpus of short texts in Russian based on Twitter messages [3], which consists of 114991 positive and 111923 negative tweets. The 10-fold cross-validation shows accuracy up to 83%.

2.4 Storing results

Every time the cron job have been executed, the new information about publication time, the amount of positive and negative messages for each federal subject is stored in the database.

2.5 Visualization

To explore temporal public mood variations and location based mood values the website was implemented. Both types of graphics were developed with the framework Google Charts [1], which provides developers with the tool for constructing interactive charts for browsers and mobile devices. There are three graphics displayed at the website. The first one is the Russia mood geo chart, where the current mood state for each federal subject is visualized. The second one and the third one are temporal mood changes plots through the day and through the week respectively.

1) Mood variations

The information about the time of the day and day of the week is extracted from messages to calculate temporal mood changes. Next, the public mood changes are calculated using the following equitation:

$$mood_t = \frac{pos_t}{pos_t + neg_t}$$

Where, pos_t is the number of positive messages in the specific period t; neg_t is the number of negative messages in the specific period t. The temporal mood changes chart through the day and through the week are plotted in the program. These charts are constructed over all data that have been processed by the program already, so the level of its accuracy and reliability increases with the number of analyzed tweets.

2) Mood geo chart

To plot the mood geo chart, for each federal subject the mood values are calculated using (7) for the last hour. Next, the federal subjects in the geo chart are marked with colors from green to red, where green color means the predominance of positive tweets; yellow color means the balance between the amount of positive and negative messages; red color means the predominance of negative tweets. Fig. 2 illustrates the example of the public mood geo chart for Russia.

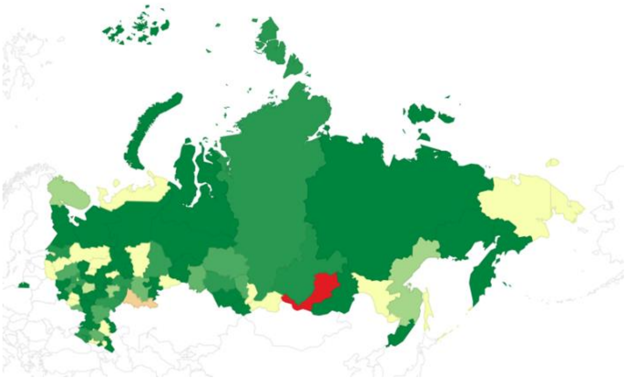


Fig. 2. Example of the public mood geo chart for Russia

3. Results

As a result, the program for public mood monitoring through Twitter content in Russian is implemented as web-service, which can be found by the URL http://twittermood-ru.appspot.com/. The program collects new messages, which are published on Twitter, in real time mode, performs sentiment analysis, process the data obtained at the previous step, and visualizes the results. The mood geo chart provides with an opportunity for monitoring mood values in different regions of Russia for the last hour. The other plots offer valuable insights about temporal public mood changes based on all collected data.

The further research will be focused on extending of analyzed feelings, that means, monitoring not only positive or negative sentiment expressions, but also the expression of fear, sadness, joy, and anger. In addition, the multiclass sentiment classification can be implemented to enhance the quality of public mood calculations.

4. Limitations

Despite a wide range of Twitter content analysis benefits, it also has some drawbacks. Technically, Twitter users are not representative of the public, consequently, tweets are not representative of the public opinion [15]. Findings in this article apply only to the population of Twitter users geo-located in the Russia. In this work, it’s possible to make claims only about the population of Russia Twitter users and not the general population.

References

[1]. “Charts | Google Developers,” *Google Developers*. [Online]. Available: https://developers.google.com/chart/. [Accessed: 18-Mar-2017].

[2]. R. Collins, D. May, N. Weinthal, and R. Wicentowski, “SWAT-CMW: Classification of Twitter Emotional Polarity using a Multiple-Classifier Decision Schema and Enhanced

Emotion Tagging," *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pp. 669–672, 2015. – 2015.

[3]. "Corpus of short texts in Russian," *Julia Rubtsova*. [Online]. Available: http://study.mokoron.com/. [Accessed: 18-Mar-2017].

[4]. "Datastore - NoSQL Schemaless Database | Google Cloud Platform," *Google Cloud Platform*. [Online]. Available: https://cloud.google.com/datastore/. [Accessed: 18-Mar-2017].

[5]. L. Dey, S. Chakraborty, A. Biswas, B. Bose, and S. Tiwari, "Sentiment Analysis of Review Datasets Using Naïve Bayes' and K-NN Classifier," *International Journal of Information Engineering and Electronic Business*, vol. 8, no. 4, pp. 54–62, Aug. 2016.

[6]. F. Dzogang, T. Lansdall-Welfare, and N. Cristianini, "Discovering Periodic Patterns in Historical News," *Plos One*, vol. 11, no. 11, 2016.

[7]. F. Dzogang, T. Lansdall-Welfare, and N. Cristianini, "Seasonal Fluctuations in Collective Mood Revealed by Wikipedia Searches and Twitter Posts," *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, 2016.

[8]. "GeoNames," *GeoNames*. [Online]. Available: http://www.geonames.org/. [Accessed: 18-Mar-2017].

[9]. "Google App Engine Documentation | App Engine Documentation | Google Cloud Platform," *Google Cloud Platform*. [Online]. Available: https://cloud.google.com/appengine/docs/. [Accessed: 18-Mar-2017].

[10]. E. Haddi, "Sentiment analysis: text, pre-processing, reader views and cross domains," dissertation, 2015.

[11]. M. Korobov, "Morphological Analyzer and Generator for Russian and Ukrainian Languages," *Communications in Computer and Information Science Analysis of Images, Social Networks and Texts*, pp. 320–332, 2015.

[12]. S. Kumar, S. Maskara, N. Chandak, and S. Goswami, "Empirical Study of Relationship between Twitter Mood and Stock Market from an Indian Context," *International Journal of Applied Information Systems*, vol. 8, no. 7, pp. 33–37, 2015.

[13]. T. Lansdall-Welfare, F. Dzogang, and N. Cristianini, "Change-Point Analysis of the Public Mood in UK Twitter during the Brexit Referendum," *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, 2016.

[14]. B. Le and H. Nguyen, "Twitter Sentiment Analysis Using Machine Learning Techniques," *Advanced Computational Methods for Knowledge Engineering Advances in Intelligent Systems and Computing*, pp. 279–289, 2015.

[15]. A. Mitchell and P. Hitlin, "Twitter reaction to events often at odds with overall public opinion," *Pew Research Center*, vol. 4, 2013.

[16]. "Natural Language Toolkit," *Natural Language Toolkit — NLTK 3.0 documentation*. [Online]. Available: http://www.nltk.org/. [Accessed: 18-Mar-2017].

[17]. "thinkBIG – Patterns in Big Data: Methods, Applications and Implications," *thinkBIG*. [Online]. Available: http://thinkbig.enm.bris.ac.uk/. [Accessed: 18-Mar-2017].

[18]. "Tweepy," *Tweepy*. [Online]. Available: http://www.tweepy.org/. [Accessed: 18-Mar-2017].

[19]. "Twitter Developer Documentation — Twitter Developers," *Twitter*. [Online]. Available: https://dev.twitter.com/docs. [Accessed: 18-Mar-2017].

[20]. Y. Wan and Q. Gao, "An Ensemble Sentiment Classification System of Twitter Data for Airline Services Analysis," *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, 2015.

Программа для мониторинга общественных настроений в России на основе сообщений из Twitter

С.И. Сметанин <sismetanin@gmail.com>
Национальный исследовательский университет «Высшая школа экономики», 101000, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. Ежедневно пользователями социальных сетей генерируются значительные объемы текстового контента, который дополнительно содержит информацию о координатах и времени публикации. Эти данные могут быть проанализированы и использованы для оценки общего состояния большой популяции пользователей с целью решения научных вопросов из широкого спектра дисциплин. В данной статье описывается разработка программы для мониторинга общественных настроений на основе анализа тональности сообщений из русскоязычного сегмента социальной сети Twitter с использованием методов машинного обучения. В разработанном программном продукте была использована многоуровневая сетевая архитектура «клиент-сервер». Написанное на Python серверное приложение собирает сообщения пользователей через Twitter API, осуществляет предварительную обработку текста, анализирует эмоциональную окраску сообщений с использованием мультиномиального наивного Байесовского классификатора и определяет их принадлежность к административно-территориальным субъектам страны. Клиентское веб-приложение визуализирует результаты анализа тональности, которые состоят из карты настроений России, где для каждого административно-территориального субъекта указывается текущий показатель настроения, а также из графиков изменения настроения в течение дня и в течение недели. В процессе разработки программного средства были задействованы облачные сервисы. Серверная часть была развернута на платформе Google App Engine, которая позволяет выполнять веб-приложения на серверах Google, то есть полностью абстрагироваться от инфраструктуры, поэтому при работе сервер не нуждается в администрировании. Данные программы хранятся в облачной базе данных Google Cloud Datastore, которая полностью интегрирована с Google App Engine.

Ключевые слова: анализ тональности; общественные настроения; социальные сети

DOI: 10.15514/ISPRAS-2017-29(4)-22

Для цитирования: Сметанин С.И. Программа для мониторинга общественных настроений в России на основе сообщений из Twitter. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 315-324 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-22

Список литературы

[1]. Charts | Google Developers. *Google Developers* (online). Доступно по ссылке: https://developers.google.com/chart/. [Дата обращения: 18.03.2017]

[2]. R. Collins, D. May, N. Weinthal, and R. Wicentowski, "SWAT-CMW: Classification of Twitter Emotional Polarity using a Multiple-Classifier Decision Schema and Enhanced Emotion Tagging," *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pp. 669–672, 2015. – 2015.

[3]. Corpus of short texts in Russian. *Julia Rubtsova*. (online). Доступно по ссылке: http://study.mokoron.com/. [Дата обращения: 18.03.2017]
[4]. Datastore - NoSQL Schemaless Database | Google Cloud Platform. *Google Cloud Platform* (online). Доступно по ссылке: https://cloud.google.com/datastore/. [Дата обращения: 18.03.2017]
[5]. L. Dey, S. Chakraborty, A. Biswas, B. Bose, and S. Tiwari, "Sentiment Analysis of Review Datasets Using Naïve Bayes' and K-NN Classifier," *International Journal of Information Engineering and Electronic Business*, vol. 8, no. 4, pp. 54–62, Aug. 2016.
[6]. F. Dzogang, T. Lansdall-Welfare, and N. Cristianini, "Discovering Periodic Patterns in Historical News," *Plos One*, vol. 11, no. 11, 2016.
[7]. F. Dzogang, T. Lansdall-Welfare, and N. Cristianini, "Seasonal Fluctuations in Collective Mood Revealed by Wikipedia Searches and Twitter Posts," *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, 2016.
[8]. GeoNames. *GeoNames* (online). Доступно по ссылке: http://www.geonames.org/.
[9]. Google App Engine Documentation | App Engine Documentation | Google Cloud Platform. *Google Cloud Platform* (online). Доступно по ссылке: https://cloud.google.com/appengine/docs/. [Дата обращения: 18.03.2017]
[10]. E. Haddi, "Sentiment analysis: text, pre-processing, reader views and cross domains," dissertation, 2015.
[11]. M. Korobov, "Morphological Analyzer and Generator for Russian and Ukrainian Languages," *Communications in Computer and Information Science Analysis of Images, Social Networks and Texts*, pp. 320–332, 2015.
[12]. S. Kumar, S. Maskara, N. Chandak, and S. Goswami, "Empirical Study of Relationship between Twitter Mood and Stock Market from an Indian Context," *International Journal of Applied Information Systems*, vol. 8, no. 7, pp. 33–37, 2015.
[13]. T. Lansdall-Welfare, F. Dzogang, and N. Cristianini, "Change-Point Analysis of the Public Mood in UK Twitter during the Brexit Referendum," *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, 2016.
[14]. B. Le and H. Nguyen, "Twitter Sentiment Analysis Using Machine Learning Techniques," *Advanced Computational Methods for Knowledge Engineering Advances in Intelligent Systems and Computing*, pp. 279–289, 2015.
[15]. A. Mitchell and P. Hitlin, "Twitter reaction to events often at odds with overall public opinion," *Pew Research Center*, vol. 4, 2013.
[16]. Natural Language Toolkit. *Natural Language Toolkit — NLTK 3.0 documentation* (online). Доступно по ссылке: http://www.nltk.org/. [Дата обращения: 18.03.2017]
[17]. thinkBIG – Patterns in Big Data: Methods, Applications and Implications. *thinkBIG*. (online). Доступно по ссылке: http://thinkbig.enm.bris.ac.uk/. [Дата обращения: 18.03.2017]
[18]. Tweepy. *Tweepy* (online). Доступно по ссылке: http://www.tweepy.org/. [Дата обращения: 18.03.2017]
[19]. Twitter Developer Documentation — Twitter Developers. *Twitter*. (online). Доступно по ссылке: https://dev.twitter.com/docs. [Дата обращения: 18.03.2017]
[20]. Y. Wan and Q. Gao, "An Ensemble Sentiment Classification System of Twitter Data for Airline Services Analysis," *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, 2015.

Narrabat — a Prototype Service for Stylish News Retelling

I.I. Dolgaleva <iidolgaleva@edu.hse.ru>
I.A. Gorshkov <iagorshkov@edu.hse.ru>
R.E. Yavorskiy <ryavorsky@hse.ru>
Faculty of Computer Science, Higher School of Economics,
20 Myasnitskaya, Moscow, 101000, Russia

Abstract. Nowadays, news portals are forced to seek new methods of engaging the audience due to the increasing competition in today's mass media. The growth in the loyalty of news service consumers may further a rise of popularity and, as a result, additional advertising revenue. Therefore, we propose the tool that is intended for stylish presenting of facts from a news feed. Its outputs are little poems that contain key facts from different news sources, based on the texts of Russian classics. The main idea of our algorithm is to use a collection of classical literature or poetry as a dictionary of style. The facts are extracted from news texts through Tomita Parser and then presented in the form similar to a sample from the collection. During our work, we tested several approaches for text generating, such as machine learning (including neural networks) and template-base method. The last method gave us the best performance, while the texts generated by the neural network are still needed to be improved. In this article, we present the current state of Narrabat, a prototype system rephrasing news we are currently working on, give examples of generated poems, and discuss some ideas for future performance improvement.

Keywords: natural language processing; information extraction; natural language generation; tomita parser; neural networks

DOI: 10.15514/ISPRAS-2017-29(4)-23

For citation: Dolgaleva I.I., Gorshkov I.A., Yavorsky R.E. Narrabat — a Prototype Service for Stylish News Retelling. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 325-336. DOI: 10.15514/ISPRAS-2017-29(4)-23

1. Introduction

1.1 The main idea

In the era of information explosion demand for news aggregation services is always high. Classical news services like Yandex News or Google News are on the market for a long time, but their format is too restricted to satisfy all potential audiences. The motivation for Narrabat, a new news service, is to retell news in a stylish way similar to the writings of great writers and poets so as to promote consumers loyalty and to increase the revenue of news portals, for instance, from contextual advertising.

The goal of the study is to develop a methodology of rewriting news texts in a specified style and to implement it as a service. To provide a new insight into retelling news, we build an architecture of Narrabat that is rather straightforward: retrieve news from the providers, extract facts, reproduce the facts in a new form. The realization of the proposed architecture might require handling two important issues. Firstly, it is necessary to process the news and extract the main information from it. At this point, it is essential to realize what kind of unstructured data will be marked as key information. Secondly, we need to generate text in a predefined style considering extracted key words.

To make precise the scope of the study, we explore the methods of retelling the news texts in more capturing manner and build a system that today has no parallel in the integrated marketing communications in news sphere.

The paper presents the current state of the retelling service implementation we are still working on. A well-established result is that we have constructed a prototype system that is capable of producing the poem from the news text. It is to be hoped that in the not too distant future, the findings of the current research will be applied to real regularly updated news feed as a service, possibly, as a chat-bot.

The plan of the paper is the following: in section 2 we present an algorithm for producing poems from the news. In section 3 the current results are presented. Finally, section IV describes the work still to be done.

1.2 Related work

Recent years have seen the rapid growth in the number of studies devoted to the extraction of information and natural language generation. Insofar as retelling news is concerned to these two subject areas, it would be wise to cover both of them in the paper.

Nowadays, state-of-the-art approaches of fact extraction go far beyond the earliest systems, where the patterns are found referring to rules of grammar [1], [2]. However, an involvement of highly qualified experts in the field or linguists is believed to be a significant drawback of these approaches. Some of them are briefly recalled in the next few paragraphs.

The next coherent idea about highlighting the facts from the text was to propose an algorithm that was able to be trained independently or ”almost independently”, namely, using active learning techniques [3], [4].

As the task of the researches became more complicated, and the need to distinguish an implicitly expressed meaning occurred, the aforementioned approaches lose its efficiency. And the researches shifted their attention to generative models [5] and conditional models [6].

Shedding light on the text generation approaches, the first things that arises is that text in natural language may be generated via predetermined rules [7], [8], when a set of templates is composed to map semantics to utterance. This approach is supposed to be conventional one. These systems are believed to be simple and easy to control, however, at the same time, no scalable due to limited number of rules, and, consequently, output texts.

Furthermore, utilization of statistical approaches in sentence planning are still based on hand-written text generators, whether choosing the most frequent derivation in context-free grammar [9] or maximizing the reward in reinforcement learning [10]. By the way, further researches are aimed at minimizing human participation and rely on learning sentence planning rules from labelled corpus of utterances [11], which also require a huge markup by linguists.

The next set of approaches in natural language generation is based on corpus-driven dependencies. The systems in this direction imply the construction of class-based n-gram language model [12] or phrase-based language model [13]. Moreover, a significant number of researchers utilize active learning in order to generate texts [14], [15].

The use of neural network-based approaches in natural language generation is still relatively unexplored. Although, there are studies that present the high-quality recurrent neural network-based language models [16], [17] that are able to model arbitrarily long dependencies. In addition, it is worth emphasizing that the usage of Long Short-term Memory (LSTM) network may try to solve the vanishing gradient problem [18] such as in [10].

2. Data and Method

2.1 The news sources

In this framework, we utilize short news texts that were extracted from Russian-language informational portal ”Yandex.News”. The collection of news consists of 330 texts on different topics, for instance, society, economy, policy, to name but a few (ultimately, 22 topics). This collection of news texts was composed of texts on diverse topics wilfully so as to consider all lexical, syntactic and morphological particularities of each of the themes in order to create universal system of text processing and generation.

Every text in the collection comprises no more than three sentences except a title. It is worth emphasizing that the format of short texts leads itself well with highlighting the main information from the text. It follows from the fact that every sentence is quite informative to extract key knowledge by means of rule-based approach.

2.2 Fact extraction

To provide basic information from the news, we propose to extract a kind of extended grammatical basis of the sentences. To that end, we use Tomita-parser [19] that allows to extract structured data (facts) from text in natural language. The tool is much more flexible and effective in key information detection and extraction than, for example, metric tf-idf since it allows to retrieve finite chains of words from all the positions in the sentence, not only successive words.

Open-source Tomita-parser, in contrast to similar non-commercial fact extraction software, accounts for specificity of work with the Russian language and has more or less detailed documentation. The tool was implemented by developers of Yandex on the basis of GLP-parser [20], which utilizes context-free grammars, dictionaries of keywords and interpreter.

To get a new insight into extracting the meaning of the texts, a dictionary (gazetteer) and grammar was compiled. As mentioned before, we suggest that the main idea of the sentence is fixed in common basis of the sentence, a kind of analogue of the grammatical basis. Given the opportunity to construct Russian-language sentences with the inversion, the grammar consists of the two main rules:

$$S \rightarrow Subject\ Predicate \mid Predicate\ Subject$$

Every non-terminal derives a string of words dependent on the root words, namely, for Subject it may be adjective and for Predicate it may be addition or adverb.

After the required string of words is found, Tomita-Parser transforms it into fact and represents it in the result collection of labelled texts, which, in turn, is prepared for text retelling.

2.3 Poems collection

To teach our system the poetry style we have used writings of Alexander Blok [21] and Nikolay Nekrasov [22] retrieved from Maksim Moshkov on-line library Lib.Ru [23]. We have chosen to utilize particularly these poets as their poems possess artistic and rhythmic harmony, and clearly traceable metrical feet. In further work, we plan to expand the collection of poetry by Agniya Barto, Athanasius Fet and Fedor Tyutchev.

2.4 Learn and produce methods

Besides the method that is described above, we tested another ways of generating word sequences, such as neural networks. For example, we trained a network with LSTM-layer which was expected to generate poems, using a huge dataset of Pushkin's poems from [24]. (LSTM for generating poems was successfully applied in [25], [26], [27]). The result we got was a bit insufficient due to low computational power of our computer and small network size. Further implementations with additional layers increased the quality of generated poems, but it is still being trained, so we are not ready yet to present its results.

Table 1 presents the example of quatrain generated by the first version of our neural networks:

Table 1. The poem produced by neural networks

Narrabat output v.00	Ко в жаме стрьк иреланье, И сталили пореланье И по почаль в сореннем По сеанно переланий.

On the Table I it could be seen that although the poem consists of non-existent Russian-language words, the strings of characters in words virtually resemble real words in their structure. The second thing to sharpen the issue addressing the table is that three out of four strings in the quatrain have the same number of syllables (while the fourth line has only one syllable less). The makings of the rhythms, as well, are evident. Given all the above, we treat the neural networks as a paramount direction for our further research.

2.5 Current version of the algorithm

Apart from training neural networks to generate poems, we are so far to seek the most conspicuously well-turned poem generator. To that end, we use template-base method described below.

First, in order to break words into syllables, we utilize an improved version of an algorithm of P. Hristov in the modification of Dymchenko and Varsanofiev [28] that comprises a set of syllabication rules that are applied sequentially. Then syllables of potentially matching subjects and predicates are compared using the following heuristic:

- The number of syllables must coincide.
- Vowels inside syllables have priority over consonants.
- The last syllable has priority over the other.

Search for the similar sentences returns pieces of classical writings, which are used then as templates for the resulting text gener-ation. The output poems ought to be sought in the Section 3.

3. Results

Below is an example produced by current release (v.01) of our Narrabat system. We start from a news description and extract subject and predicate, see Table 2.

Table 2. Original news

Source	Общегородской субботник пройдет в следующую субботу, 15 апреля.

Extracted basis	Subject = общегородской субботник
	Predicate = пройдет

The same is done for all sentences in the collection, see example in Table 3.

Table 3. Original piece from the collections

Source	А виноградные пустыни, Дома и люди — всё гроба. Лишь медь торжественной латыни Поет на плитах, как труба.

Extracted basis	Subject = медь торжественной латыни
	Predicate = поет

The implemented similarity measure allows us to figure out that the subjects and the predicates are quite similar, see Table 4. Notice the same number of syllables and almost identical endings.

Table 4. Example of a similar pairs match

Subjects	Predicates
медь тор-жест-вен-ной ла-ты-ни	по-ет
об-ще-го-род-ской суб-бот-ник	прой-дет

Now we can replace the matching pairs, see Table 5 for an example of the resulting poem.

Table 5. The final result of the algorithm

Narrabat output v.00	А виноградные пустыни, Дома и люди — всё гроба. Лишь общегородской субботник Пройдет на плитах, как труба.

One can see that the resulting text keeps subject and predicate from the original fact and at the same time the inserted fragment smoothly fits the style of the poem and do not destroy its structure.

All readers are able to have a closer look at the details of implementation of our Narrabat system and access the source code that is open and available on GitHub [29].

4. Conclusion and future research directions

In the paper we have proposed a prototype of system that is capable of retelling the news as poems that resembles style of great writers.

In the course of the work we discovered that the collection of poems have to be drastically enlarged in order to generate high-quality poems. Given the exploration, Nikolay Nekrasov has demonstrated more mapping potential in our tasks as he wrote more common sentences than Alexander Blok.

Moreover, the aforementioned metrics of mapping the subjects and predicates from news and poems does not cover all cases to be universal, for instance, in further releases it may take into account rhyme explicitly. Although the first results presented above are somehow promising, still a lot is on the to do list:

- Improve the quality fact extraction by extending the parsing rules.
- Use available dictionaries of accentuation to take into account the rhythmic structure of a sentence.
- Apply machine learning techniques to better grasp the style of a sample writing.
- Extend the algorithm to cover other parts of sentences, namely, objects and complements.

References

[1]. Douglas E Appelt, Jerry R Hobbs, John Bear, David Israel, and Mabry Tyson. Fastus: A finite-state processor for information extraction from real-world text. In IJCAI, volume 93, pages 1172–1178, 1993.

[2]. R Mooney. Relational learning of pattern-match rules for information extraction. In Proceedings of the Sixteenth National Conference on Artificial Intelligence, volume 328, page 334, 1999.

[3]. François Mairesse, Milica Gašić, Filip Jurčíček, Simon Keizer, Blaise Thomson, Kai Yu, and Steve Young. Phrase-based statistical language generation using graphical models and active learning. In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 1552–1561. Association for Computational Linguistics, 2010.

[4]. Aidan Finn and Nicolas Kushmerick. Active learning selection strategies for information extraction. In Proceedings of the International Workshop on Adaptive Text Extraction and Mining (ATEM-03), pages 18–25, 2003.

[5]. Kristie Seymore, Andrew McCallum, and Roni Rosenfeld. Learning hidden markov model structure for information extrac-tion. In AAAI-99 workshop on machine learning for information extraction, pages 37–42, 1999.

[6]. Adwait Ratnaparkhi. Learning to parse natural language with maximum entropy models. Machine learning, 34(1-3):151– 175, 1999.

[7]. Adam Cheyer and Didier Guzzoni. Method and apparatus for building an intelligent automated assistant, March 18 2014. US Patent 8,677,377.

[8]. Hugo Gonçalo Oliveira and Amílcar Cardoso. Poetry generation with poetryme. In Computational Creativity Research: Towards Creative Machines, pages 243–266. Springer, 2015.

[9]. Anja Belz. Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models. Natural Language Engineering, 14(04):431–455, 2008.

[10]. Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. arXiv preprint arXiv:1508.01745, 2015.

[11]. Amanda Stent and Martin Molina. Evaluating automatic extraction of rules for sentence plan construction. In Proceedings of the SIGDIAL 2009 Conference: The 10th Annual Meeting of the Special Interest Group on Discourse and Dialogue, pages 290–297. Association for Computational Linguistics, 2009.

[12]. Adwait Ratnaparkhi. Trainable approaches to surface natural language generation and their application to conversational dialog systems. Computer Speech & Language, 16(3):435–455, 2002.

[13]. François Mairesse and Steve Young. Stochastic language generation in dialogue using factored language models. Compu-tational Linguistics, 2014.

[14]. Gabor Angeli, Percy Liang, and Dan Klein. A simple domain-independent probabilistic approach to generation. In Pro-ceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, pages 502–512. Association for Computational Linguistics, 2010.

[15]. Ravi Kondadadi, Blake Howald, and Frank Schilder. A statistical nlg framework for aggregated planning and realization. In ACL (1), pages 1406–1415, 2013.

[16]. Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on, pages 5528–5531. IEEE, 2011.

[17]. Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In Interspeech, volume 2, page 3, 2010.

[18]. Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2):157–166, 1994.

[19]. Yandex LLC. Tomita-parser tool to extract structured data from texts. https://tech.yandex.ru/tomita/. Accessed: 2017-04-10.

[20]. Masaru Tomita. Lr parsers for natural languages. In Proceedings of the 10th International Conference on Computational Linguistics and 22nd annual meeting on Association for Computational Linguistics, pages 354–357. Association for Computational Linguistics, 1984.

[21]. Alexander Blok. Sobranie sochinenij v 8 tomah [Collected Works in 8 volumes]. Gosudarstvennoe izdatel’stvo hudozhestvennoj literatury [State Publishing House of Fiction], Moscow, 1960-1963 (in Russian).

[22]. Nikolaj Nekrasov. Polnoe sobranie stihotvorenij N. A. Nekrasova v 2 tomah [Complete collection of poems by N.A. Nekrasov in 2 volumes]. Tipografija A. S. Suvorina [Printing house of AS Suvorin], Sankt-Peterburg, 1899 (in Russian).

[23]. Lib.ru: Library of Maksim Moshkov. http://lib.ru/. Accessed: 2017-04-10.

[24]. Alexander Pushkin. Sobranie sochinenij v desyati tomah. Tom vtoroj. Stihotvoreniya 1823-1836 [Collected works in ten volumes. Volume 2. Poems of 1823-1836]. [State Publishing House of Fiction], Moscow, 1959—1962 (in Russian).

[25]. Anna Rumshisky Peter Potash, Alexey Romanov. Ghostwriter: Using an lstm for automatic rap lyric generation. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1919–1924, 2015.

[26]. Rui Yan. i, poet: Automatic poetry composition through recurrent neural networks with iterative polishing schema. In Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16), pages 2238–2244, 2016.

[27]. Yejin Choi Marjan Ghazvininejad, Xing Shi and Kevin Knight. Generating topical poetry. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1183–1191, 2016.

[28]. Yura Batora. Algorithm for splitting words into syllables. https://sites.google.com/site/foliantapp/project-updates/hyphenation. Accessed: 2017-04-10.

[29]. Rostislav Yavorskiy Irina Dolgaleva, Ilya Gorshkov. Narrabat. https://github.com/onobot/allbots/tree. Accessed: 2017-04-10.

Narrabat — прототип сервиса для пересказа новостей в формате стихотворений

И.И. Долгалева <iidolgaleva@edu.hse.ru>
И.А. Горшков <iagorshkov@edu.hse.ru>
Р.Э. Яворский <ryavorsky@hse.ru>
Факультет компьютерных наук, Высшая Школа Экономики,
101000, Россия, Москва, ул. Мясницкая, 20

Аннотация. В интернете все большую популярность приобретают СМИ, отказывающиеся от общепринятого формального способа изложения новостей и делающие акцент на креативности предоставляемого контента. Яркими примерами могут послужить паблик "Лентач" из социальной сети "ВКонтакте", сопровождающий каждую новость мемами, и ресурс "КАКТАМ?", оборачивающий заголовки в намеренно сверхэмоциональную форму. Мы решили реализовать инструмент Narrabat, пересказывающий новости в еще одном необычном стиле. Его задача - преобразовывать новостные ленты, взятые из сторонних источников, в небольшие стихотворения, отражающие ключевые события новостных сюжетов. В качестве основы для генерации стихов используется большая коллекция русской классики (состоящая из, к примеру, произведений Блока и Некрасова). Одним из главных достоинств выбранной нами формы пересказа и созданного инструмента в частности является то, что, при всей оригинальности вывода, процесс его генерации полностью автоматизирован, в отличие от сервисов, описанных выше. Инструмент работает в несколько этапов: сначала происходит выделение фактов из заголовков выгруженных новостей при помощи Tomita Parser, после чего факты передаются в модуль, отвечающий за генерацию стихотворения. По ходу работы мы использовали несколько подходов для генерации стихотворений, такие, как алгоритмы, построенные на правилах, и машинное обучение, включая нейронные сети. На данном этапе наилучший результат дал первый метод, однако работа по обучению нейронной сети ведется до сих пор. В данной статье мы опишем текущие результаты работы, приведем примеры сгенерированных стихотворений, а также перечислим направления для дальнейшего улучшения инструмента.

Ключевые слова: обработка естественного языка; извлечение информации; генерация текста; томита парсер; нейронные сети

DOI: 10.15514/ISPRAS-2017-29(4)-23

Для цитирования: Долгалева И.И., Горшков И.А., Яворский Р.Э. Narrabat — прототип сервиса для пересказа новостей в формате стихотворений. *Труды ИСП РАН*, том 29, вып. 4, 2017 г., стр. 325-336 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(4)-23

Список литературы

[1]. Douglas E Appelt, Jerry R Hobbs, John Bear, David Israel, and Mabry Tyson. Fastus: A finite-state processor for information extraction from real-world text. In IJCAI, volume 93, pages 1172–1178, 1993.

[2]. R Mooney. Relational learning of pattern-match rules for information extraction. In Proceedings of the Sixteenth National Conference on Artificial Intelligence, volume 328, page 334, 1999.

[3]. François Mairesse, Milica Gašić, Filip Jurčíček, Simon Keizer, Blaise Thomson, Kai Yu, and Steve Young. Phrase-based statistical language generation using graphical models and active learning. In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 1552–1561. Association for Computational Linguistics, 2010.

[4]. Aidan Finn and Nicolas Kushmerick. Active learning selection strategies for information extraction. In Proceedings of the International Workshop on Adaptive Text Extraction and Mining (ATEM-03), pages 18–25, 2003.

[5]. Kristie Seymore, Andrew McCallum, and Roni Rosenfeld. Learning hidden markov model structure for information extrac-tion. In AAAI-99 workshop on machine learning for information extraction, pages 37–42, 1999.

[6]. Adwait Ratnaparkhi. Learning to parse natural language with maximum entropy models. Machine learning, 34(1-3):151– 175, 1999.

[7]. Adam Cheyer and Didier Guzzoni. Method and apparatus for building an intelligent automated assistant, March 18 2014. US Patent 8,677,377.

[8]. Hugo Gonçalo Oliveira and Amílcar Cardoso. Poetry generation with poetryme. In Computational Creativity Research: Towards Creative Machines, pages 243–266. Springer, 2015.

[9]. Anja Belz. Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models. Natural Language Engineering, 14(04):431–455, 2008.

[10]. Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. arXiv preprint arXiv:1508.01745, 2015.

[11]. Amanda Stent and Martin Molina. Evaluating automatic extraction of rules for sentence plan construction. In Proceedings of the SIGDIAL 2009 Conference: The 10th Annual Meeting of the Special Interest Group on Discourse and Dialogue, pages 290–297. Association for Computational Linguistics, 2009.

[12]. Adwait Ratnaparkhi. Trainable approaches to surface natural language generation and their application to conversational dialog systems. Computer Speech & Language, 16(3):435–455, 2002.

[13]. François Mairesse and Steve Young. Stochastic language generation in dialogue using factored language models. Compu-tational Linguistics, 2014.

[14]. Gabor Angeli, Percy Liang, and Dan Klein. A simple domain-independent probabilistic approach to generation. In Pro-ceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, pages 502–512. Association for Computational Linguistics, 2010.

[15]. Ravi Kondadadi, Blake Howald, and Frank Schilder. A statistical nlg framework for aggregated planning and realization. In ACL (1), pages 1406–1415, 2013.

[16]. Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on, pages 5528–5531. IEEE, 2011.

[17]. Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In Interspeech, volume 2, page 3, 2010.

[18]. Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2):157–166, 1994.

[19]. Yandex LLC. Томита-парсер. https://tech.yandex.ru/tomita/. Дата обращения 10.04.2017.

[20]. Masaru Tomita. Lr parsers for natural languages. In Proceedings of the 10th International Conference on Computational Linguistics and 22nd annual meeting on Association for Computational Linguistics, pages 354–357. Association for Com-putational Linguistics, 1984.

[21]. Александр Блок. Собрание сочинений в 8 томах. Государственное издательство художественной литературы, Москва, 1960-1963.

[22]. Николай Некрасов. Полное собрание стихотворений Н.А. Некрасова в 2 томах. Типография А. С. Суворина, Санкт-Петербург, 1899.

[23]. Lib.ru: Библиотека Максима Мошкова. http://lib.ru/. Дата обращения 10.04.2017. .

[24]. Александр Пушкин. Собрание сочинений в десяти томах. Том второй. Стихотворения 1823-1836. 1823-1836.

[25]. Anna Rumshisky, Peter Potash, Alexey Romanov. Ghostwriter: Using an lstm for automatic rap lyric generation. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1919–1924, 2015.

[26]. Rui Yan. i, poet: Automatic poetry composition through recurrent neural networks with iterative polishing schema. In Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16), pages 2238–2244, 2016.

[27]. Yejin Choi, Marjan Ghazvininejad, Xing Shi and Kevin Knight. Generating topical poetry. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1183–1191, 2016.

[28]. Yura Batora. Algorithm for splitting words into syllables. https://sites.google.com/site/foliantapp/project-updates/hyphenation. Дата обращения 10.04.2017.

[29]. Rostislav Yavorskiy, Irina Dolgaleva, Ilya Gorshkov. Narrabat. https://github.com/onobot/allbots/tree. Дата обращения 10.04.2017.