ИСП РАН

# ТРУДЫ ИНСТИТУТА СИСТЕМНОГО ПРОГРАММИРОВАНИЯ РАН

# PROCEEDINGS OF THE INSTITUTE FOR SYSTEM PROGRAMMING OF THE RAS

ISSN PRINT
2079-8156
ТОМ 31
ВЫПУСК 5

ISSN ONLINE
2220-6426
VOLUME 31
ISSUE 5

ИНСТИТУТ СИСТЕМНОГО ПРОГРАММИРОВАНИЯ
ИМ. В.П. ИВАННИКОВА РАН

МОСКВА, 2019

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

**DOI: 10.15514/ISPRAS-2019-31(5)-1**

# Автоматизированное тестирование фронтенда транслятора TCG для Qemu

[1]*Д.С. Колтунов, ORCID: 0000-0003-4556-9463 <koltunov@ispras.ru>*
[1]*В.Ю. Ефимов, ORCID: 0000-0003-3433-6787 <real@ispras.ru>*
[1,2]*В.А. Падарян, ORCID: 0000-0001-7962-9677 <vartan@ispras.ru>*

[1]*Институт системного программирования им. В.П. Иванникова 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*

**Аннотация.** Реализация новой виртуальной процессорной архитектуры в Qemu предполагает создание фронтенда динамического двоичного транслятора TCG для данной процессорной архитектуры. Существующие на сегодняшний день системы тестирования фронтенда TCG используют подход на основе сравнения с эталоном той же процессорной архитектуры. В качестве эталона могут выступать реальный процессор, виртуальная машина с большей точностью эмуляции или другая реализация двоичного транслятора. Однако не всегда такие эталоны доступны, зачастую они могут вообще не существовать. Данная работа нацелена на тестирование реализации процессорной архитектуры в Qemu в условиях отсутствия необходимого эталона для сравнения. Предлагаемый подход основывается на том, что даже для малораспространённой процессорной архитектуры, как правило, доступен пакет `binutils` и компилятор языка Си. Си-программа может одинаково выполняться на различных процессорных архитектурах, если удаётся избегать в ней ситуаций неопределенного или реализационно зависимого поведения. Это позволяет проводить сравнение хода работы двух разных исполняемых файлов на тестируемой виртуальной машине и машине разработчика. Объектами сравнения такого подхода выступают сущности языка Си, на котором разрабатываются тесты. Подход реализован в программном средстве c2t (CPU Testing Tool) и входит в состав программного комплекса автоматизации разработки моделей устройств и вычислительных машин для Qemu, исходный код которого доступен по адресу https://github.com/ispras/qdt. c2t реализовано на языке программирования Python, поддерживает тестирование Qemu в режиме полносистемной эмуляции и в режиме эмуляции уровня пользователя. Данное средство пригодно как для тестирования фронтендов TCG, полученных с использованием системы автоматизации создания фронтендов TCG, так и реализованных классическим способом (вручную).

**Ключевые слова:** Qemu; автоматизированное тестирование фронтенда TCG; QDT; GDB RSP

**Для цитирования:** Колтунов Д.С., Ефимов В.Ю., Падарян В.А. Автоматизированное тестирование фронтенда транслятора TCG для Qemu. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 7–24. DOI: 10.15514/ISPRAS–2019–31(5)–1

**Благодарности:** Работа поддержана грантом РФФИ № 16-29-09632

# Automated testing of a TCG frontend for Qemu

[1]*D.S. Koltunov, ORCID: 0000-0003-4556-9463 <koltunov@ispras.ru>*
[1]*V.Y. Efimov, ORCID: 0000-0003-3433-6787 <real@ispras.ru>*
[1,2]*V.A. Padaryan, ORCID: 0000-0001-7962-9677 <vartan@ispras.ru>*

[1]*Ivannikov 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, Russian Federation*

**Abstract.** Implementing a new target architecture in Qemu involves creation of a dynamic binary translator TCG front-end for that architecture. Testing is necessary to verify correctness of that translator component. Currently, existing TCG front-end testing systems use an approach based on a comparison with an oracle. Such oracle have the same processor architecture. And an oracle may be a real processor, a high-fidelity emulator or another binary translator. Unfortunately, such oracles are not always available. This paper is devoted to testing a target architecture implementation in Qemu when the necessary oracle is not available. The main idea is following. There is observation, a program written in a high-level programming language is expected to execute equally regardless of processor architecture. In other words, one can use a real processor with a different architecture for comparison. In this paper, it is the processor of a developer AMD64 machine. The comparison objects are the term of a high-level programming language. I.e. tests are written in C. C language was chosen for this purpose, because, on the one hand, it is fairly close to the hardware, and, on the other, it has compilers for many processor architectures. The approach is implemented in CPU Testing Tool (c2t) which is part of QDT. Source code is available at https://github.com/ispras/qdt. The tool is implemented in Python programming language and supports testing of Qemu in both full system and user level emulation modes. c2t is suitable for testing TCG front-ends which are generated by the automatic TCG front-end generation system or implemented in the classical way (manually).

**Keywords:** Qemu; automated testing of a TCG frontend; QDT; GDB RSP

**For citation:** Koltunov D.S., Efimov V.Y., Padaryan V.A. Automated testing of a TCG frontend for Qemu. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019 г., pp. 7-24 (in Russian). DOI: 10.15514/ISPRAS–2019–31(5)–1

**Acknowledgements.** This work was supported by RFBR grant № 16-29-09632

## *1. Введение*

Виртуальные вычислительные машины применяются для решения разнообразных задач, включая исследования в рамках информационной безопасности. Одной из таких задач является динамический анализ, который нуждается в контролируемой среде выполнения исследуемого машинного кода. Удобный и распространённый способ реализации контролируемой среды – программный эмулятор, поскольку он обеспечивает изоляцию средств анализа от анализируемого кода. На практике такой способ хорошо подходит для исследования компьютерных вирусов и другого вредоносного ПО.

Эмулятор Qemu наиболее удобен для этой цели, поскольку обладает рядом полезных свойств: полностью открытый исходный код (лицензия GPL), поддержка разнообразных гостевых архитектур (Intel x86, AMD 64, ARM, MIPS, PowerPC, SPARC и др.), реализация важных, с точки зрения динамического анализа, технологий и возможностей [1].

Тем не менее, при необходимости провести динамический анализ для малораспространенной процессорной архитектуры, скорее всего, аналитик столкнется с тем, что готовой виртуальной машины найти не удастся. В этом случае придётся решать задачу разработки виртуальной машины, в которую входит большая подзадача реализации процессорной архитектуры.

Реализация новой процессорной архитектуры в Qemu, кроме прочего, предполагает создание фронтенда (англ. frontend) динамического двоичного транслятора TCG для данной процессорной архитектуры. Как правило, фронтенды TCG создаются вручную, но на данный момент уже есть средства, предлагающие некоторую автоматизацию [2]. В любом случае для проверки корректности фронтенда транслятора необходимо тестирование. Оно является неотъемлемым этапом разработки ПО, который своевременно сокращает количество допущенных программистом ошибок, что, в свою очередь, повышает точность эмуляции и качество анализа исследуемого кода.

Самым распространённым способом тестирования реализации процессорной архитектуры в Qemu является сравнение с эталоном («test oracle») [3]. Чтобы провести тестирование таким способом необходима доступность эталона, а также возможность получать от него информацию о состоянии выполняющейся на нем программы. Существующие на сегодняшний день системы тестирования фронтенда TCG используют в качестве эталона реальный процессор, виртуальную машину с большей точностью эмуляции или альтернативную реализацию двоичного транслятора. Оценивается корректность работы отдельной инструкции, которая выполняется в тестируемом и эталонном окружениях, подготовленных специальным образом. После ее выполнения сравниваются состояния заданных регистров процессора и содержимого памяти в эмуляторе и эталоне.

Однако не всегда есть возможность работы с эталоном. Это обусловлено недоступностью необходимого реального оборудования, отсутствием у оборудования отладочных интерфейсов, недоступностью альтернативной виртуальной машины или двоичного транслятора нужной процессорной архитектуры.

Целью данной работы является разработка подхода к тестированию реализации процессорной архитектуры в Qemu в условиях отсутствия необходимого эталона для сравнения и последующая реализация соответствующего программного средства, позволяющего проводить автоматизированное тестирование.

Предлагаемый подход основывается на том, что даже для малораспространенной процессорной архитектуры будет доступен некоторый набор программных инструментов разработки: пакет `binutils` или его аналог, компилятор языка Си. Располагая такими инструментами можно проводить тестирование, не опускаясь на уровень отдельных команд, а выполняя тестовые Си-программы и сравнивая их поведение в терминах языка Си –на уровне значений переменных. Один и тот же тест компилируется для двух различных машин: Qemu (тестируемая процессорная архитектура) и машины (используемой в качестве эталона), на которой ведётся разработка.

В данной работе сделано следующее:

- разработан набор тестов;
- разработан способ оценки достигнутого покрытия кода;
- реализовано средство автоматизации тестирования;
- проведены эксперименты.

## *2. Обзор родственных работ*

Существующие на сегодняшний день системы тестирования реализации процессорной архитектуры в Qemu основываются на сравнении с эталоном данной процессорной архитектуры. Они делятся на следующие типы.

- Системы, которые направлены на обнаружение неточностей реализации процессорной архитектуры x86 с целью повысить прозрачность с точки зрения динамического анализа. В качестве эталона для сравнения выступают реальный

процессор или виртуальная машина с большей точностью эмуляции. Такие системы описаны в работах EmuFuzzer [4], KEmuFuzzer [5] и PokeEMU [6].

- Системы генерации тестов для тестирования реализации не только процессорной архитектуры x86. Эталоном является реальный процессор. К данному типу относится системы RISU [7] и MicroTESK [8].
- Системы, использующие в качестве эталона для сравнения другую реализацию двоичного транслятора. Такой системой является MeanDiff [9].

Далее кратко будет рассмотрена каждая из этих систем.

## 2.1 EmuFuzzer, KEmuFuzzer и PokeEMU

EmuFuzzer –прототип, в котором реализована полностью автоматизированная методика тестирования для эмуляторов процессора (Qemu, Valgrind, Pin и BOCHS), основанная на фаззинге. Данная методика может быть использована для автоматического обнаружения расхождений конфигурации среды (т.е. состояние регистров процессора и содержимое памяти) в эмулируемом и физическом процессорах. Для тестирования эмулятора создаётся большое количество тестов, которые запускаются на эмулируемом и реальном процессорах. В конце выполнения каждого теста сравниваются конфигурации двух сред. Любое расхождение является признаком неправильного поведения эмулятора.

KEmuFuzzer –прототип, который реализует автоматизированную методику (как и EmuFuzzer) для тестирования четырёх современных виртуальных машин: BOCHS, Qemu, VirtualBox и VMware. Отличие от EmuFuzzer в том, что тестируются полносистемные эмуляторы, и что в качестве эталона выступает KVM.

PokeEMU –инструмент тестирования эмулятора. Он автоматически генерирует наборы тестов с большим покрытием и сравнивает поведение эмулятора с реальной машиной, выполняя тесты на обоих из них. Инструмент генерирует тестовые сценарии с помощью символьного выполнения. Данным инструментом был протестирован Qemu. PokeEMU также использует в качестве эталона KVM.

Имеются ещё работы подобного типа [10] и [11].

## 2.2 RISU и MicroTESK

Инструмент RISU (Random Instruction Sequences for Userspace) –инструмент для проверки точности реализации набора инструкций процессора в таких виртуальных машинах, как Valgrind и Qemu. Он состоит из двух частей.

- Генератор, который выводит случайный машинный код на основе входного файла, описывающего шаблоны набора инструкций.
- Тестовая программа, которая запускает сгенерированный код, как на тестируемом, так и на реальном оборудовании.

Инструмент может использоваться для тестирования эмуляторов, предназначенных для запуска пользовательских приложений, например, Valgrind и Qemu linux-user. Для архитектуры ARM RISU поддерживает тестирование 32-битных наборов команд A32 (ARM) и T32 (Thumb), а также 64-битного набора команд A64. Инструмент также поддерживает архитектуры PPC и m68k.

Инструмент MicroTESK (Microprocessor TEsting and Specification Kit) –среда генерации тестовых программ на языке ассемблера целевой процессорной архитектуры для функциональной верификации микропроцессоров. Также данный инструмент предоставляет возможность автоматизированного конструирования генераторов тестов на основе формальных спецификаций необходимой процессорной архитектуры. Такие

тесты можно запускать в Qemu и на реальном процессоре с целью выявления расхождений в их поведении.

## 2.3 MeanDiff

MeanDiff –инструмент тестирования промежуточных представлений для двоичных трансляторов. Он реализует идею поиска семантических ошибок во фронтендах (binary lifters) существующих двоичных трансляторов.

Подход к тестированию основывается на сравнении семантики промежуточных представлений трансляторов. Разработано некоторое специальное унифицированное промежуточное представление, которое используется для единообразного описания семантики разных промежуточных представлений.

Процесс тестирования заключается в том, что на вход сравниваемым двоичным трансляторам подаётся некоторый машинный код. Этот код переводится трансляторами в своё промежуточное представление. Затем все полученные промежуточные представления транслируются в унифицированное промежуточное представление. После этого результаты сравниваются между собой, и производится проверка на наличие семантических расхождений. Данным инструментом можно протестировать правильность реализаций фронтендов двоичных трансляторов, которые имеются в Qemu, Valgrind, BINSEC и т.д.

## 2.4 Выводы

В рассмотренных работах при тестировании реализации процессорной архитектуры в Qemu используются разные виды эталонов: реальный процессор, виртуальная машина с большей точностью эмуляции, а также другая реализация двоичного динамического транслятора. Но все они той же процессорной архитектуры, что и тестируемая реализация. Однако не всегда есть доступ к таким эталонам. Для решения этой проблемы в данной работе предлагается подход, опирающийся на альтернативный, но всегда доступный эталон.

# *3. Предлагаемый подход к тестированию*

Предлагаемый в данной работе подход к тестированию фронтендов TCG различных процессорных, подобно другим подходам, использует сравнение с эталоном. Но эталоном будет выступать процессор *любой* архитектуры (к какому-нибудь имеется доступ всегда). Главная проблема заключается в том, у процессора другой архитектуры отличаются регистры (длина, номенклатура, количество, и пр.), варианты типовых команд, длина адреса и прочие архитектурные особенности. Но обычно процессоры используются для выполнения алгоритмов, логика которых не привязана к конкретной архитектуре. Например, вычисление арифметического среднего, сортировка, поиск в массиве и т.д.

Независимо от архитектуры процессора такой алгоритм на каждом своём шаге должен находиться в состоянии, зависящем только от входных данных. Суть предлагаемого подхода заключается в повышения уровня представления состояния регистров процессора и памяти до абстрактных понятий, используемых в алгоритме. Затем предлагается сравнивать путь выполнения и состояния высокоуровневых абстракций (таких как переменные).

Этого предлагается достичь за счет написания тестовых сценариев на языке программирования высокого уровня, что дает следующие возможности для тестирования.

- Возможность использовать в качестве эталона для сравнения –персональный компьютер.

- Повторное использование тестов. Уход от машинной зависимости тестов, которые в существующих решениях пишутся на языке ассемблера или генерируются в виде машинного кода, предоставляет возможность использовать один раз написанный тест для тестирования фронтендов TCG различных архитектур.

Очевидно, что у данного подхода есть много ограничений, которые рассмотрены ниже. Тем не менее показано, что так можно протестировать значительную часть реализации фронтенда.

Наиболее подходящим для решения данной задачи является использование высокоуровневого языка программирования Си.

Объектами сравнения являются значения переменных и номера строк программы. Стоит отметить, что для проверки корректности функционирования фронтенда TCG сравнение значений переменных и позиции выполнения является практически достаточным для большинства случаев. Архитектурные особенности основной и целевой системы отличаются, но логика работы с переменными у них одинаковая. Немаловажным является то, что предложенный подход к тестированию автоматизируется.

Таким образом, предлагаемый подход заключается в компиляции программы (теста), написанной на языке Си, под тестируемую архитектуру и архитектуру основной машины с последующим запуском в Qemu и ОС пользователя под контролем отладчика.

## 3.1 Ограничения и тонкости

В стандарте языка Си говорится о ситуациях, когда поведение некоторых конструкций языка может быть различным в зависимости от платформы и реализации компилятора (undefined behaviour и implementation-defined behavior). Если аккуратно избежать попадания в данные ситуации, то скомпилированная программа на Си должна одинаково выполняться независимо от процессорной архитектуры и компилятора. При этом будем считать, что процессор и компилятор реализованы без ошибок, так как вероятность наличия в них ошибок много меньше, чем в тестируемой реализации процессорной архитектуры.

При составлении тестовых программ необходимо учитывать и другие тонкости. Для того, чтобы компилятор не устранил переменную как избыточную при оптимизации, нужно использовать для данной переменной квалификатор типа `volatile`. Другим примером является то, что часто при программировании микроконтроллеров тип `int` языка Си имеет диапазон значений (размер), который отличается от того диапазона, к которому привыкли Си-программисты под IA32 и AMD64. В данном случае рекомендуется использовать подходящий тип из `<stdint.h>`. Например, `int32_t` или `int16_t` – в зависимости от того, какой диапазон значений на самом деле нужен. При этом, явное указание диапазона значений переменной может влиять на выбор компилятором определённых вариантов инструкций, а также поможет проверить правильность реализации в эмуляторе поведения при переполнении.

Не всегда возможно добиться от компилятора использования некоторых инструкций. В некоторых случаях от разработчика тестов требуется определённая изобретательность. В крайнем случае, интересующую инструкцию можно внедрить ассемблерной вставкой, а при помощи препроцессора добиться, чтобы для эталона вместо этой инструкции выполнялся эквивалентный Си-код. Другими словами, подход не исключает классический способ тестирования с написанием тестов на ассемблере. Он позволяет сэкономить время в тех случаях, когда требуемого покрытия можно добиться, используя Си. Такая гибридизация является предметом дальнейших исследований.

Кроме того, процессоры часто обладают инструкциями, семантика которых не описывается языком Си. Например, инструкция остановки процессора до следующего

аппаратного прерывания (`hlt` в i386) или векторные инструкции (расширение SSE для x86). Тестирование таких инструкций выходит за рамки данной работы.

Таким образом, подход главным образом ориентирован на проверку правильности реализации:

- арифметики – сложение, вычитание, умножение, деление;
- побитовых операций – умножение, сложение, сложение по модулю 2, инверсия, сдвиги;
- обработки данных и операций с памятью – операции с регистрами, стеком и т.д.;
- операций изменения потока управления – вызовы подпрограмм, условная и безусловная передача управления, возврат управления.

Создание тестов, которые обеспечивают наибольшее покрытие по инструкциям, реализованных в тестируемом фронтенде TCG, является отдельной сложной задачей и выходит за рамки данной работы. Однако, ряд показательных тестов всё-таки был реализован для оценки подхода.

## *4. Реализация автоматизированного тестирования*

Разработанный подход был реализован в программном средстве автоматизированного тестирования фронтендов TCG –c2t (CPU Testing Tool), которое входит в состав программного комплекса автоматизации разработки моделей устройств и вычислительных машин для Qemu [1]. c2t написан на языке Python. Исходный код QDT (в том числе и c2t) является открытым.

Подход предполагает использование отладчика для сравнения значений переменных и пути выполнения (номеров строк).

Получение состояний переменных обеспечивается с помощью модуля отладки. Помимо контроля за выполнением отладчик интерпретирует низкоуровневые данные в терминах языка Си, так как эмулятор и эталон оперируют понятиями памяти и регистра процессора. Подробно об этом рассказано выше.

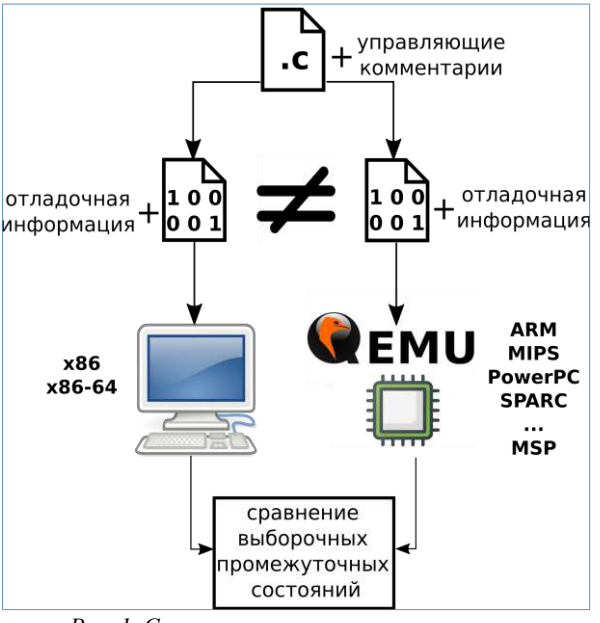


*Рис. 1. Схема осуществления тестирования*
*Fig. 1. Testing approach*

При сравнении пути выполнения программы на каждом шаге сравнивается не счётчик команд, как это происходит в EmuFuzzer и KEmuFuzzer, а позиция в коде тестового сценария (номер строки). Разумеется, ни эмулятор, ни эталон не могут предоставить такую высокоуровневую информацию: им доступно только значение своего счётчика команд. Поэтому тут тоже задействован модуль отладки.

Так как сравнение состояний всех переменных после выполнения каждой строки теста в ходе проведения тестирования является избыточным, то необходим механизм указания определённых строк и имён переменных для сравнения. Другими словами: механизм тонкой настройки тестирования. Данный механизм реализован путём добавления специальных управляющих комментариев напротив необходимых строк исходного кода теста (см. рис. 1).

Сам факт наличия управляющего комментария к строке означает, что на данной строке будет приостановлено выполнение теста эталоном и тестируемой виртуальной машиной, и будут получены состояния переменных для дальнейшего сравнения. Другими словами, комментарии указывают c2t те места, на которые необходимо поставить точки останова, а также имена переменных, значения которых необходимо проверить.

В Qemu реализован сервер внешней отладки гостевого кода по протоколу удалённой отладки GDB (GDB Remote Serial Protocol), основными возможностями которого являются:

- чтение/запись значений регистров процессора;
- чтение/запись в оперативную память;
- установка точек останова по управлению (breakpoint);
- установка точек останова по обращению к памяти (watchpoint).

Это означает, что для отладки гостевого кода может быть использован отладчик, поддерживающий данный протокол. Например, отладчик GDB. Однако использование отладчика GDB для проведения тестирования фронтенда Qemu усложняет процесс подготовки к проведению тестирования. Это обусловлено высокой трудоёмкостью процесса модификации отладчика GDB, как и любого другого отладчика, ориентированного на взаимодействие с человеком и не являющегося программной библиотекой. Таким образом, необходимо использовать средство отладки, предоставляющее достаточную гибкость.

В качестве решения данной проблемы выступает отладочный API QDT. QDT является инструментом автоматизации разработки моделей устройств и вычислительных машин для Qemu, написанный на языке программирования Python. Отладочный API QDT использует модуль «pyrsp» [12] с открытым исходным кодом, также написанный на языке Python. Модуль «pyrsp» реализует API для взаимодействия по протоколу RSP. Также отладочный API QDT использует модуль «pyelftools» [13] для работы с отладочной информацией в формате DWARF. На основе данного API и реализовано программное средство автоматизированного тестирования фронтенда транслятора Qemu: c2t.

Средство c2t поддерживает тестирование Qemu в режиме полносистемной эмуляции и в режиме эмуляции уровня пользователя. Обычно при тестировании TCG виртуальные устройства не представляют интереса. Более того, их деятельность может мешать. Т.е. поддержки эмуляции уровня пользователя обычно бывает достаточно. Но бывают случаи, когда для работы алгоритма высокого уровня нужно некоторое устройство.

Например, в микроконтроллерах может отсутствовать инструкция умножения. Вместо неё присутствует устройство-умножитель, не входящее в АЛУ процессора, а работающее как периферийное устройство. Для поддержки реализации таких архитектур применяется полносистемная версия Qemu. Подобный случай, строго говоря, входит за рамки

тестирование фронтенда TCG. Однако необходимость в проверке реализации подобных нюансов работы остаётся.

Ниже представлена схема проведения тестирования с использованием разработанного программного средства (см. рис. 2).

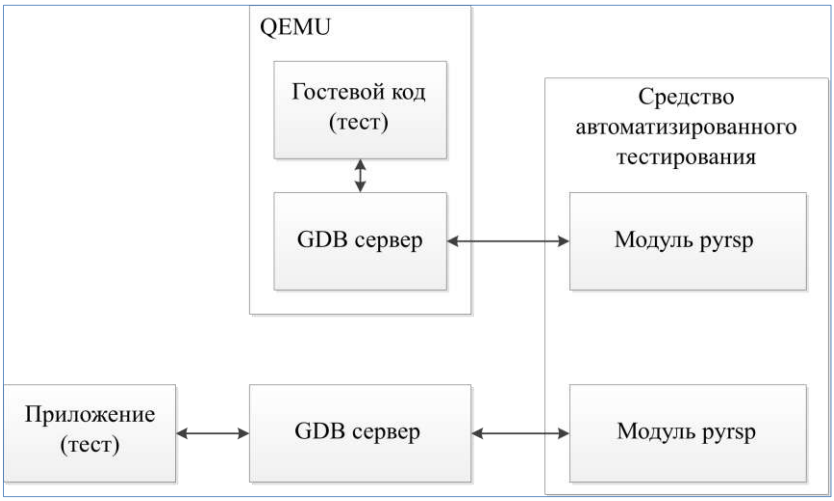


*Рис. 2. Схема проведения тестирования*
*Fig. 2. Testing scheme*

Конфигурация тестирования реализуется путём добавления управляющих комментариев в исходном коде теста, которые являются выражениями на языке Python, выполняемыми штатными средствами в окружении, настроенном нужным образом. Данные комментарии пишутся человеком, проводящим тестирование, и содержат команды специального вида, задающие проверки, которые будут выполнены во время тестирования. На данный момент программным средством поддерживаются следующие команды:

- `br` –установка точки останова для однократной проверки соответствия позиций выполнения тестового сценария;
- `brc` –установка точки останова для многократной проверки соответствия позиций выполнения тестового сценария;
- `bre` –завершить выполнение теста;
- `ch` –установка точки останова для однократной проверки соответствия позиций выполнения и равенства значений переменных тестового сценария;
- `chc` –установка точки останова для многократной проверки соответствия позиций выполнения и равенства значений переменных тестового сценарияю.

Пример теста и конфигурации тестирования, с использованием управляющих комментариев, представлен на рис. 3. Для поддержки тестирования в Qemu необходимо добавить код, предоставляющий возможность отладчику читать и писать в регистры процессора, а также отвечающий за поддержку установки отладочных точек останова (вставка транслятором необходимого кода tcg, реализующего точки останова).

Поддержка RSP фронтендом TCG не является обязательным (в техническом смысле) при реализации процессорной архитектуры Qemu. С практической точки зрения, RSP реализуют всегда. При этом её сравнительно просто добавить. Следовательно, будем считать, что поддержка RSP имеется всегда.

```
int main(void) {
    volatile uint32_t a = 0xABCDEF, b = 0x12345678, c = 0, i;

    for(i = 0; i < 20; i++) {
        if(i % 2) {
            // однократная проверка позиций выполнения
            c = b - a; //$br
        }
        else {
            // многократная проверка позиций выполнения
            c = b + a; //$brc
        }

        // однократная проверка значений всех переменных
        c = b & a; //$ch

        // однократная проверка значений переменной `c`
        c = b | a; //$ch.c

        // многократная проверка значений переменной `c`
        c = b ^ a; //$chc.c

        /* Также есть возможность комбинировать команды:
          - однократная проверка позиций выполнения и
            значений переменных `a` и `b`
          - многократная проверка значений переменной `c`
        */
        c = 0; //$br, chc.c, ch.a, ch.b
    }

    // завершить тестирование
    return 0; //$bre
}
```

*Рис. 3. Листинг теста с управляющими комментариями*
*Fig. 3. Test listing*

Для поддержки тестирования в Qemu необходимо добавить код, предоставляющий возможность отладчику читать и писать в регистры процессора, а также отвечающий за поддержку установки отладочных точек останова (вставка транслятором необходимого кода tcg, реализующего точки останова). Поддержка RSP фронтендом TCG не является обязательным (в техническом смысле) при реализации процессорной архитектуры Qemu. С практической точки зрения, RSP реализуют всегда. При этом её сравнительно просто добавить. Следовательно, будем считать, что поддержка RSP имеется всегда.

Программное средство c2t поддерживает многопоточный режим. Несколько тестов могут выполняться параллельно. Также оно имеет режим вывода промежуточной информации в ходе проведения тестирования. На момент написания данной работы реализовано 145 тестов. Названия тестов, которые тестировщику необходимо запустить, задаются регулярными выражениями.

## *5. Описание экспериментального стенда*

В качестве эталона использовался персональный компьютер на базе процессора i7-4790 архитектуры AMD64. Версии ПО, использованного в эксперименте приведены в табл. 1. Звёздочкой помечены реализации, находящиеся в официальной ветке «master».

*Табл. 1. Версии использованного ПО*
*Table 1. Versions of used software*

| фронтенд | Qemu | GCC |
|---|---|---|
| ARM | 2.5.0* | 4.9.3 |
| MIPS | 2.5.0* | 5.4.0 |
| MSP430_1 | 2.6.9 | 7.3.2 |
| MSP430_2 | 2.12.1 | 7.3.2 |

Покрытие было оценено `gcov` версии 5.4.0.

## *6. Оценка покрытия*

Фронтенд транслятора TCG реализован на языке Си. Реализация фронтенда TCG представляет собой набор вложенных конструкций `switch` (синтаксический анализатор), ветки которых содержат семантики соответствующих машинных инструкций. Данные семантики выражаются на промежуточном представлении TCG. Генерация кода TCG осуществляется путём вызова специальных функций.

В данной работе было проведена оценка двух покрытий. Под первым покрытием подразумевается количество выполненных (покрытых) ветвей конструкций `switch` фронтенда TCG, соответствующих уникальным инструкциям гостевого процессора. Оценка данного покрытия была произведена с использованием утилиты `gcov` для исследования покрытия кода. Данная утилита входит в состав пакета GCC. Под вторым покрытием подразумевается количество выполненных Qemu уникальных инструкций гостевого процессора.

Для оценки данного покрытия Qemu запускалась в режиме пошагового выполнения (`-singlestep`), с включенной отладочной трассировкой и выключенными сцеплением блоков транслированного кода (`-d exec,nochain`).

Перед пояснением обозначенных параметров напомним, что TCG транслирует гостевой блок блоками, получая подпрограммы для процессора основной машины. Выполнение подпрограммы приводит к изменениям в гостевой машине, которые бы произошли в ней от выполнения блока исходного гостевого кода, исполняй она его непосредственно. Посоле выполнения каждой подпрограммы происходит возврат в служебный код эмулятора с целью поиска/трансляции следующего блока (а именно, соответствующей ему подпрограммы). Если для данного блока известен следующий блок, то Qemu добавляет в конец подпрограммы переход на точку входа в следующую подпрограмму, **минуя возврат в служебный код**. Эта оптимизация, ускоряющая работу эмулятора, называется *сцеплением* блоков.

В режиме пошагового выполнения в блок всегда попадает одна гостевая инструкция. Перед выполнением подпрограммы **служебный код** эмулятора пишет в трассу **гостевой** адрес точки входа в блок. Из-за сцепления блоков, предотвращающим возврата в служебный код, в трассу не попадают гостевые адреса инструкций в прицепленных блоков. Это мешает учёту покрытых инструкций, поэтому сцепление блоков было отключено при оценке покрытия.

Гостевой адрес в трассе позволяет автоматически с помощью дизассемблера определить множество покрытых гостевых инструкций.

Далее приводится разъяснение того, какие инструкции при оценке покрытия считаются разными. Очевидно, что инструкции сложения и умножения всегда можно назвать разными. Но часто инструкции, выполняющие одну операцию, поддерживают разные режимы адресации операндов:

- непосредственное значение;
- регистр (так называемая «прямая адресация»);
- ячейка памяти, адресуемая значением в регистре (так называемая «косвенная адресация») и др.

Архитектура MSP430 имеет 7 режимов адресации. Даже если семантика инструкции не меняется при изменении режима адресации одного из операндов, то всё равно используется другой код из реализации фронтенда. А разработчику важно покрыть весь код фронтенда. Вариативность не ограничивается режимами адресации и всегда зависит от конкретной архитектуры процессора. Таким образом, критерий одинаковости инструкций зависит как от архитектуры, так и от решаемой разработчиком задачи. Далее описываются критерии, которых придерживались авторы для протестированных в данной работе гостевых архитектур.

## 6.1 Классификация инструкций

Инструкции получаются из 145 тестов на Си, имеющихся на момент написания работы (они входят в состав c2t). Данные тесты содержат следующие операции, предоставляемые языком Си:

1. присваивание (=);
2. арифметические: сложение (+), вычитание (−), умножение (*) и деление (/);
3. побитовые умножение (&), сложение (|), сложение по модулю 2 (^), отрицание (~), сдвиги вправо (>>)/влево (<<);
4. сравнения: равенство (==), неравенство (!=), больше (>), меньше (<), больше или равно (>=), меньше или равно (<=).

Также имеются тесты, которые содержат условные конструкции, циклы, вызовы функций. Данными тестами можно проверить реализации инструкций передачи и возврата управления, а также работу со стеком. Для последнего используются тесты, содержащие вызовы функций с большим количеством параметров. Это гарантируется размещением входных параметров в стеке, т.к. многие соглашения о вызове стараются передавать параметры через регистры.

Чтобы проверить реализации различных вариаций одной и той же инструкции процессора, тесты выполнены в вариантах, содержащих 8, 16, 32 и 64 разрядные знаковые и беззнаковые переменные.

Для подтверждения работоспособности (применимости) предложенного в рамках данной работы подхода было произведено тестирование существующих в официальном репозитории Qemu реализаций процессорных архитектур ARM32 и MIPS32, а также ещё двух реализаций процессорной архитектуры MSP430 (далее MSP430_1 и MSP430_2) , не находящихся в официальном репозитории. Тесты для данных архитектур были скомпилированы соответствующим компилятором `gcc` с выключенными оптимизациями (`-O0`). Реализация MSP430_1 доступна на github [14]. Однако для того, чтобы протестировать данную реализацию, потребовалось внести в неё поддержку RSP, согласно требованиям, обозначенным выше. Также потребовалось сократить количество тестов, пропустив проверку умножения. Это сделано по причине того, что в архитектуре MSP430 умножение реализуется аппаратно: на микроконтроллере присутствует специальное периферийное устройство. Модель же соответствующего устройства в эмуляторе пока отсутствует.

Реализация MSP430_2 является собственной реализацией, в которой были умышленно допущены ошибки с целью выявления их программным средством тестирования. Все эти ошибки были успешно обнаружены. Также были обнаружены и неумышленные ошибки.

Инструкции процессора можно разбить на следующие классы:

1. обработка данных и операции с памятью;
2. арифметические и логические операции;
3. операции изменения потока управления;
4. операции, специфичные для процессора.

Как было сказано выше, тестирование реализации инструкций 4 класса выходит за рамки данной работы и, следовательно, их нецелесообразно включать в оценку покрытия. Инструкции классов с 1 по 3 входят в оценку покрытия.

## 6.2 Покрытые инструкции

В табл. 2 содержатся полученные результаты по покрытию тестированием ветвей фронтенда транслятора TCG. Представлено отношение количества покрытых ветвей конструкций `switch` от общего количества ветвей, содержащих семантику уникальных инструкций. Данный результат также соответствует отношению количества выполненных Qemu инструкций от общего количества инструкций тестируемой гостевой процессорной архитектуры, реализованных в Qemu. В подсчёт общего количества инструкций для каждой из таких процессорных архитектур вошли уникальные по семантике инструкции 1, 2 и 3 классов. Те инструкции, которые возможно покрыть тестами, имеющимися на момент написания работы. Из полученного результата исключено число инструкций, которые предназначены для удобства написания программ и которые реализуются ассемблером.

Примерами таких инструкций для архитектуры ARM32 являются `adr` (загрузить адрес в регистр, реализуется инструкцией `add` или `sub`), `push` (положить в стек, реализуется инструкцией `stm`) и `pop` (достать из стека, реализуется инструкцией `ldm`). Для архитектуры MIPS32: `li` (загрузить «непосредственное» значение в регистр, реализуется инструкциями `lui` и `ori`) и `move` (копировать значение регистра в регистр, реализуется инструкцией `or`). Для архитектуры MSP430: `pop` (достать из стека, реализуется инструкцией `mov` со специальным режимом адресации), `inc` (инкрементировать, реализуется инструкцией `add`) и `ret` (возврат из подпрограммы, реализуется инструкцией `mov`).

Кроме того, все различные вариации отдельно взятой инструкции при подсчёте рассматриваются как одна инструкция. Например, для архитектуры ARM32 в качестве таких инструкций выступают инструкции с условным выполнением (например, `addge`), а также обновляющие значения флагов (например, `adds`). Не вошли в подсчет и специфичные для конкретного процессора инструкции. Информация для подсчёта общего количества инструкций была взята из [15], [16] и [17].

*Табл. 2. Количество покрытых инструкций*
*Table 2. Number of covered instructions*

| Фронтенд | Покрыто | Всего | % |
|---|---|---|---|
| ARM32 | 38 | 61 | 62 |
| MIPS32 | 42 | 67 | 63 |
| MSP430_1 | 21 | 27 | 78 |
| MSP430_2 | 12 | 19 | 63 |

В соответствии с табл. 2, в среднем покрывается около 60% инструкций. Такое покрытие не является выдающимся результатом. Однако данное покрытие возможно достичь, приложив небольшие усилия. Необходимо добавить поддержку процессорной архитектуры в c2t и установить компилятор для данной архитектуры.

Тем не менее разработанный инструмент предоставляет возможность покрыть инструкции, о которых было сказано выше. Например, инструкцию `addge` (сложить, если истинно условие «больше или равно») можно покрыть кодом, показанным на рис. 4.

```
void main(void)
{
    volatile int32_t a = 0xa, b = 0xb, c;
    if (a >= b)
        c = a + b;
    return;
}
```

*Рис. 4. Листинг теста, покрывающего инструкцию `addge`*
*Fig. 4. Test listing covering `addge` instruction*

При этом необходимо скомпилировать этот код, включив оптимизацию размера кода (параметр `-Os` для компилятора `gcc`). Это также актуально и для некоторых других архитектур, имеющих инструкции, которые выполняются, если истинно условие. Создание тестов, покрывающих различные вариации отдельных инструкций, является архитектурно зависимой задачей и выходит за рамки данной работы.

*Табл. 3. Список покрытых/не покрытых инструкций*
*Table 3. List of covered/non-covered instructions*

| Фронтенд | Класс | Покрытые инструкции | Не покрытые инструкции |
|---|---|---|---|
| ARM32 | 1 | ldr, mov, str, stm, ldm, mvn | — |
| | 2 | sub, lsl, lsr, eor, cmp, rsb, asr, add, adc, and, mul, orr, tst, mla, sbc, rrx, rsc, umull, ror | smull, smlal, umlal, cmn, teq, bic |
| | 3 | b, bl, bmi, beq, bcs, ble, blt, bhi, bne, bls, bcc, bgt, bge | bpl, bvs, bvc, bl(cc) |
| MIPS32 | 1 | movn, lb, lw, slt, sltiu, slti, sltu, lbu, mflo, mfhi, lui, sw, lhu, sh, sb | movz, mthi, mtlo, ll, lh, lwl, lwr, swl, swr, sc |
| | 2 | addu, addiu, subu, mul, multu, mult, div, divu, nor, sllv, srlv, and, sll, xor, srav, ori, andi, sra, srl, or | add, addi, sub, clo, clz, madd, maddu, msub, msubu, xori |
| | 3 | bltz, bgtz, jal, bne, beq, jr, bgezal | blez, bgez, bltzal, j, jalr |
| MSP430_1 | 1 | mov, push, sxt | swpb |
| | 2 | add, addc, sub, subc, cmp, and, bis, xor, rra, rrc, bit bic | dadd |
| | 3 | call, jnz, jge, jc, jeq, jl | reti, jmp, jn, jnc |
| MSP430_2 | 1 | mov | — |
| | 2 | add, sub, cmp, and, bis, xor | bit, addc, bic |
| | 3 | call, jnz, jeq, jge, jl | jc, jmp, jn, jnc |

Реализация архитектуры ARM32 была протестирована путём запуска тестов на виртуальных процессорах cortex-m3 и arm926. Реализация архитектуры MIPS32 была протестирована путём запуска тестов на виртуальных процессорах r4000 и 4kc.

Для тестирования ARM32 и MIPS32 были использованы все 145 тестов, для MSP430_1 – 69 тестов, а для MSP430_2 –49 тестов.

В табл. 3 представлены все инструкции, покрытые и непокрытые тестированием, с разбиением их по классам. Для процессорной архитектуры ARM32 запись `bl(cc)` в непокрытых инструкциях 3 класса является сокращённой записью 14 инструкций. Она подразумевает все условные вариации инструкции `bl`.

Приведенные в табл. 3 данные позволяют сделать вывод, что для всех протестированных в данной работе архитектур покрыты основные арифметические и логические инструкции, инструкции обработки данных и работы с памятью, инструкции изменения потока управления. Не покрыты некоторые инструкции условного перехода. Это обусловлено тем, что в языке Си нет выражений, имеющих семантику соответствующих инструкций. Некоторые инструкции не покрыты по причине отсутствия необходимых тестов. Например, для архитектуры ARM32 не покрыта инструкция `bic` (очистка определённых битов).

## *7. Направления дальнейших исследований*

Перспективными направлениями дальнейших исследований являются следующие.

- Исследование влияния применения оптимизаций компилятора на покрытие инструкций и сам подход. Необходимо оценить, как будет улучшаться покрытие кода. Также необходимо использовать при написании тестов различные приемы, снижающие негативное влияние оптимизаций. Например, оптимизации могут «перемешивать» отображения строк исходного кода на строки машинного кода, в таком случае необходимо использовать `asm volatile`.
- Создание тестов с наибольшим покрытием инструкций, а также автоматизация данного процесса.
- Способы принуждения компилятора к использованию определённых гостевых инструкций.

## *8. Заключение*

В ходе данной работы был проведён анализ существующих решений тестирования фронтенда транслятора TCG эмулятора Qemu. Было установлено, что рассмотренные решения используют для сравнения эталон той же процессорной архитектуры, что и у тестируемой реализации. Данное требование делает неприменимыми эти решения при отсутствии такого эталона.

В работе предложен подход к тестированию, который направлен на решение указанной ы Идея предлагаемого подхода заключается в том, что можно написать программу на языке высокого уровня так, что её получится скомпилировать под разные процессорные архитектуры. Тем самым достигается возможность использовать фиксированный эталон – процессор машины разработчика. Однако накладывается ограничение на покрытие реализаций инструкций, специфичных для конкретной процессорной архитектуры. На основе данного подхода реализовано программное средство автоматизированного тестирования.

Применимость подхода была подтверждена тестированием реализаций двух процессорных архитектур (ARM32 и MIPS32), находящихся в официальном репозитории Qemu, а также ещё двух реализаций (MSP430_1 и MSP430_2), не находящихся в официальном репозитории. В собственной реализации MSP430_2 были обнаружены как умышленно оставленные ошибки, так и не умышленные ошибки.

Также в работе приведена оценка покрытия инструкций процессорных архитектур для протестированных реализаций. Согласно полученному результату в среднем покрывается около 60% инструкций. Такое покрытие не является выдающимся результатом. Однако данное покрытие возможно достичь, приложив небольшие усилия. Предложенным подходом удалось покрыть основные арифметические и логические инструкции, инструкции обработки данных и работы с памятью, инструкции изменения потока управления.

## Список литературы / References


[1]. Ефимов В.Ю., Беззубиков А.А., Богомолов Д.А., Горемыкин О.В., Падарян В.А. Автоматизация разработки моделей устройств и вычислительных машин для QEMU. Труды ИСП РАН, том 29, вып. 6, 2017 г., стр. 77-104 / Efimov V.Yu., Bezzubikov A.A., Bogomolov D.A., Goremykin O.V., Padaryan V.A. Automation of device and machine development for QEMU. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 6, 2017, pp. 77-104 (In Russian). DOI: 10.15514/ISPRAS-2017-29(6)-4.

[2]. Bezzubikov A., Belov N., Batuzov K. Automatic dynamic binary translator generation from instruction set description. In Proc. of the 2017 Ivannikov ISPRAS Open Conference, 2017, pp. 27-33. DOI: 10.1109/ISPRAS.2017.00012.

[3]. W.E. Howden, Theoretical and empirical studies of program testing. In Proc. of the 3rd international conference on Software engineering, 1978, pp. 305-311.

[4]. Lorenzo Martignoni, Roberto Paleari, Giampaolo Fresi Roglia, Danilo Bruschi. Testing CPU emulators. In Proc. of the 18th international symposium on Software testing and analysis, 2009, pp. 261-272.

[5]. Lorenzo Martignoni, Roberto Paleari, Giampaolo Fresi Roglia, Danilo Bruschi. Testing system virtual machines. In Proc. of the 19th international symposium on Software testing and analysis, 2010, pp. 171-182.

[6]. Qiuchen Yan and Stephen McCamant. Fast PokeEMU: Scaling Generated Instruction Tests Using Aggregation and State Chaining. In Proc. of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, 2018, 13 p..

[7]. Risu: random instruction sequence tester for userspace [online] Available at: https://git.linaro.org/people/pmaydell/risu.git/about/, accessed: 09.08.2019.

[8]. A.S. Kamkin, T.I. Sergeeva, S.A. Smolov, A.D. Tatarnikov, M.M. Chupilko. Extensible environment for test program generation for microprocessors. Programming and Computer Software, vol. 40, issue 1, 2014, pp 1-9.

[9]. Soomin Kim, Markus Faerevaag, Minkyu Jung, SeungIl Jung, DongYeop Oh, JongHyup Lee, Sang Kil Cha. Testing intermediate representations for binary analysis. In Proc. of the 32nd IEEE/ACM International Conference on Automated Software Engineering, 2017, pp. 353-364.

[10]. L. Martignoni, S. McCamant, P. Poosankam, D. Song, and P. Maniatis. Path-exploration lifting: Hi-fi tests for lo-fi emulators. In Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems, 2012, pp. 337-348.

[11]. Hao Shi, Abdulla Alwabel, and Jelena Mirkovic. Cardinal pill testing of system virtual machines. In Proceedings of the 23rd USENIX Security Symposium, 2014, pp. 271-285.

[12]. pyrsp. Available at: https://github.com/stef/pyrsp, accessed: 02.08.2019.

[13]. pyelftools. Available at: https://github.com/eliben/pyelftools, accessed:23.09.2018.

[14]. Qemu MSP430. Available at: https://github.com/draperlaboratory/qemu-msp, accessed: 16.07.2019.

[15]. ARM and Thumb-2 Instruction Set Quick Reference Card. Available at: http://infocenter.arm.com/help/topic/com.arm.doc.qrc0001m/QRC0001_UAL.pdf, accessed: 16.07.2019.

[16]. MIPS Instruction Reference. Available at: https://s3-eu-west-1.amazonaws.com/downloads-mips/documents/MD00565-2B-MIPS32-Q RC-01.01.pdf, accessed: 16.07.2019.

[17]. MSP430x2xx Family User's Guide. Available at: http://www.ti.com/lit/ug/slau144j/slau144j.pdf, accessed: 16.07.2019.

## Информация об авторах / Information about authors

Дмитрий Сергеевич КОЛТУНОВ – стажер-исследователь ИСП РАН. Научные интересы: анализ бинарного кода, эмуляция и виртуализация.

Dmitry Sergeevich KOLTUNOV – a research trainee at ISP RAS. Research interests: binary code analysis, emulation and virtualization.

Василий Юрьевич ЕФИМОВ является младшим научным сотрудником ИСП РАН. Его научные интересы включают компиляторные технологии, безопасность ПО, анализ бинарного кода, параллельное программирование, эмуляция и виртуализация.

Vasily Yuryevich EFIMOV is a junior researcher at ISP RAS. His research interests include compiler technologies, software security, binary code analysis, parallel programming, emulation and virtualization.

Вартан Андроникович ПАДАРЯН является кандидатом физико-математических наук, ведущим научным сотрудником ИСП РАН, доцентом кафедры системного программирования ф-та ВМК МГУ. Его научные интересы включают компиляторные технологии, безопасность ПО, анализ бинарного кода, параллельное программирование, эмуляция и виртуализация.

Vartan Andronikovich PADARYAN is Candidate of Physical and Mathematical Sciences, Leading Researcher at ISP RAS, Associate Professor of the System Programming Department of the The faculty of Computational Mathematics and Cybernetics of Lomonosov Moscow State University. His research interests include compiler technologies, software security, binary code analysis, parallel programming, emulation and virtualization.

**DOI: 10.15514/ISPRAS-2019-31(5)-2**



# Интроспекция конфигурации периферийных устройств эмулятора QEMU

*Н.И. Фурсова, ORCID: 0000-0001-6817-4670 <natalia.fursova@ispras.ru>*
*П.М. Довгалюк, ORCID: 0000-0003-2483-5718 <pavel.dovgaluk@ispras.ru>*

*Новгородский Государственный университет имени Ярослава Мудрого*
*173003, Россия, Великий Новгород, ул. Санкт-Петербургская, д.41*

**Аннотация.** QEMU – широко используемый и достаточно точный эмулятор, способный эмулировать десятки гостевых систем. Эмуляция системы предполагает настройку виртуальных устройств, которые в большом количестве поддержаны в QEMU, что влечет за собой очень длинную и запутанную строку запуска эмулятора. При использовании детерминированного воспроизведения ситуация усложняется не только дополнительными и не вполне очевидными параметрами, но и необходимостью синхронизации строк запуска записи и воспроизведения. Машины могут иметь разный набор устройств в зависимости от платформы и даже версии эмулятора. В статье рассматривается получение информации об устройствах эмулятора QEMU через QEMU Machine Protocol для использования этих данных в графическом интерфейсе. Графический интерфейс QemuGUI поддерживает полный цикл работы с эмулятором: создание и настройка виртуальной машины, запуск в обычном режиме и в режимах детерминированного воспроизведения, взаимодействие с машиной через монитор QEMU.

**Ключевые слова:** QEMU; GUI; QMP; графический интерфейс; эмулятор

Для цитирования: Фурсова Н.И., Довгалюк П.М. Интроспекция конфигурации периферийных устройств эмулятора QEMU. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 25-36. DOI: 10.15514/ISPRAS-2019-31(5)-2

**Благодарности:** работа выполнена при поддержке гранта РФФИ (18-07-00900А) и в рамках выполнения государственного задания Минобрнауки России № 2.6146.2017/8.9.

## Introspection of QEMU emulator peripherals configuration

*N.I. Fursova, ORCID: 0000-0001-6817-4670 <natalia.fursova@ispras.ru>*
*P.M. Dovgalyuk, ORCID: 0000-0003-2483-5718 <pavel.dovgaluk@ispras.ru>*

*Yaroslav-the-Wise Novgorod State University*
*41, Sankt-Peterburgskaya st., Velikiy Novgorod, 173003, Russia*

**Abstract.** QEMU is a widely used and fairly accurate emulator capable of emulating dozens of guest systems. Emulation of the system involves the configuration of virtual devices, which are supported in large numbers in QEMU, which entails a very long and complicated command line to start the emulator. When using deterministic replay, the situation is complicated not only by additional and not quite obvious parameters, but also by the need to synchronize recording and retrace launch command lines. Machines can have a different set of devices depending on the platform and even the version of the emulator. The article describes obtaining information about the devices of the QEMU emulator through the QEMU Machine Protocol for using this data in a graphical interface. The QemuGUI graphical interface supports the full cycle of work with the emulator: creating and configuring a virtual machine, starting in normal mode and in deterministic replay mode, interacting with the machine through a QEMU monitor.

**Keywords:** QEMU;, GUI; QMP; emulator

**For citation:** Fursova N.I., Dovgalyuk P.M. Introspection of QEMU emulator peripherals configuration. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019, pp. 25-36 (in Russian). DOI: 10.15514/ISPRAS-2019-31(5)-2

**Acknowledgements.** The work was partially supported by the Ministry of Education and Science of Russia, research project No. 2.6146.2017/8.9, and by the Russian Foundation of Basic Research (research grant 18–07–00900 A).

## *1. Введение*

Эмулятор QEMU [1, 2] является одним из передовых эмуляторов для разработчиков средств анализа кода. Интересен он тем, что является программным обеспечением с открытым исходным кодом и поддерживает большое количество архитектур гостевых систем.

В связи с тем, что QEMU эмулирует аппаратное обеспечение довольно точно, он позволяет указывать огромное количество настроек, описывающих устройства машины, которые необходимо передавать в командную строку запуска. Настройки эти разной степени сложности, а командная строка может раздуваться до невероятных размеров [3, 4].

Отдельно следует сказать о работе с детерминированным воспроизведением [5]. Детерминированное воспроизведение – это технология, позволяющая записывать и многократно воспроизводить сценарии работы эмулятора. При использовании детерминированного воспроизведения появляются дополнительные трудности, например, каждое блочное устройство должно быть оснащено прослойкой дополнительных параметров, что не всегда очевидно и также раздувает командную строку. Кроме того, пользователь может иметь необходимость пользоваться разными сборками эмулятора, что вынуждает его или постоянно заглядывать в командную строку (если он ее сохранил для многоразового использования) или поддерживать файлы журналов в строгом порядке.

Пример командной строки для записи сценария:

```
D:\\Program\\Qemu\\qemu-system-x86_64w.exe -m 512 -drive
file=F:\\temp.qcow2,if=none,id=img-direct0 -drive
driver=blkreplay,if=none,image=img-direct0,id=img-blkreplay0 -device
ide-hd,drive=img-blkreplay0,bus=ide.0 -icount
shift=7,rr=record,rrfile=F:/projects/1,rrsnapshot=first-
snapshot,rrperiod=5 -net none -qmp tcp:127.0.0.1:5500,server,nowait -
vnc 127.0.0.1:0 -drive
file=f:/DEBUG_VS2012x86/FV/ovmf.fd,if=pflash,snapshot=on -machine q35
-machine smm=on -vga std
```

То же самое, но без записи сценария:

```
D:\\Program\\Qemu\\qemu-system-x86_64w.exe -m 512 -hda D:\\image.qcow2
-net none -qmp tcp:127.0.0.1:5500,server,nowait -vnc 127.0.0.1:0 -
drive file=D:/longpath/ovmf.fd,if=pflash,snapshot=on -machine q35 -
machine smm=on -vga std
```

Запутаться в необходимых настройках может даже опытный пользователь, а найти ошибку – зачастую задача непростая.

Необходимо синхронизировать командные строки запусков записи и воспроизведения и удостовериться что при многократном запуске воспроизведения строка будет одинаковой, ведь в этом случае от перестановки мест слагаемых сумма может поменяться, например, если не указать шины, на которых будут устройства, и поменять местами устройства, QEMU распределит их самостоятельно и воспроизведение не сможет работать. Важно обеспечить детерминированность этого процесса.

Кроме того, справочный раздел QEMU (`-help`) также впечатляет своим размером и найти необходимую опцию не всегда легко и просто.

Все это вынуждает пользователей сохранять командные строки для запуска эмулятора для многоразового использования.

Отчасти именно сложность командной строки при использовании детерминированного воспроизведения сподвигла к разработке графического интерфейса для эмулятора QEMU – QemuGUI.

Цель QemuGUI – обеспечивать получение актуальной информации об устройствах QEMU и конструировать работоспособную командную строку. Под актуальной информацией понимается конфигурация устройств эмулятора и их свойства, полученные максимально автоматически для конкретного используемого эмулятора. Проблема заключается в том, что получить эти данные не является тривиальной задачей.

В статье рассматриваются возможности взаимодействия с QEMU для получения информации о доступных устройствах, а также предлагается графический интерфейс для работы с эмулятором.

## *2. Модель оборудования эмулятора QEMU*

QEMU предоставляет огромный набор устройств. На рис. 1 представлен пример как устройства организованы в эмуляторе.

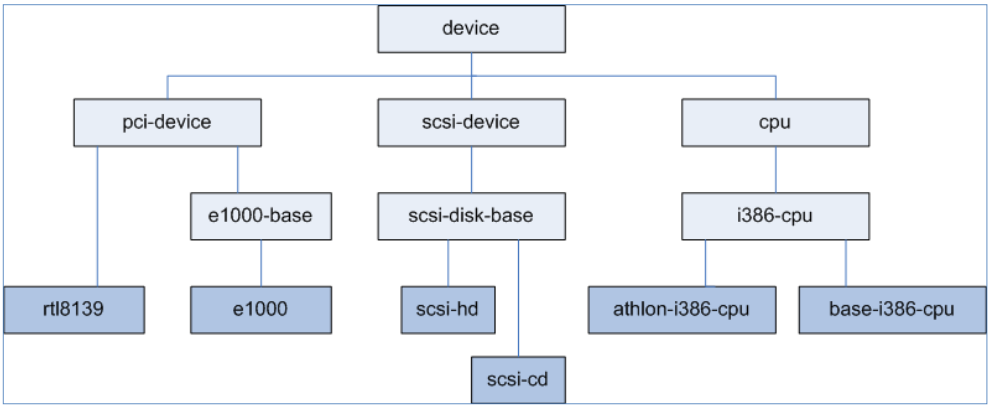


*Рис. 1. Фрагмент дерева устройств QEMU*
*Fig. 1. The part of QEMU devices tree*

Мы хотим получать список устройств и их характеристики, установленные по умолчанию, а также те, которые могут быть добавлены в эту конкретную машину.

Модель устройств должна включать шины, на которые могут быть определены устройства, сами устройства и их свойства. Она будет иметь древовидную структуру.

Примером конечного результата может послужить вывод команды монитора QEMU «`info qtree`». Ниже приведен небольшой фрагмент.

```
(qemu) info qtree
bus: main-system-bus
   type System
   dev: i440FX-pcihost, id ""
      pci-hole64-size = 2147483648 (2 GiB)
      short_root_bus = 0 (0x0)
      x-pci-hole64-fix = false
      bus: pci.0
         type PCI
         dev: e1000, id ""
```

```
        mac = "52:54:00:12:34:56"
        vlan = <null>
        ...
        netdev = "device-47"
  ...
  dev:...
```

Эта команда позволяет получить информацию обо всех устройствах, установленных на машине в данный момент. Большая часть устройств определена по умолчанию. В целом, такие данные позволили бы получить желаемую структуру устройств, но использование команды монитора не является хорошим решением, потому что вывод ответа ориентирован на пользователя, подвержен изменениям со стороны разработчика, а его разбор может быть осуществлен только на жестких сравнениях слов, используемых в выводе.

Кроме того, перебрать все возможные конфигурации машин невозможно, и какие-то устройства или шины можно упустить. Да и в целом решение получится крайне неудачным с архитектурной точки зрения. Для таких целей необходимо использовать форматы данных, предусмотренные для машинной обработки, например JSON, но для QMP (разд. 3) подобной команды нет.

## *3. Взаимодействие с QEMU*

QEMU предоставляет два канала связи – монитор [6, 7] и QEMU Machine Protocol (QMP) [8]. Монитор предназначен для пользователей, позволяет вводить команды в привычном человеку виде и получать такие же ответы. Через QMP пользователь тоже может отправлять запросы эмулятору, однако они задаются в формате JSON [9] и больше предназначены для взаимодействия между программами.

Пример простой команды остановки машины:

- монитор: `stop`
- QMP: `{"execute" : "stop"}`

И монитор, и QMP предоставляют обширный набор команд для получения информации об эмуляторе и его состоянии. QMP это машинный протокол, поэтому в работе используется он.

Команды QMP делятся на несколько групп, в нашем случае интерес представляют команды из группы запросов (query-commands) и команды управления состоянием эмулятора.

## *4. Применение QMP для получения информации об устройствах виртуальной машины*

QEMU эмулирует большое количество оборудования, которое может меняться от версии к версии или от настроек машины. Важно иметь актуальные списки моделей устройств, доступных в машине, поэтому вариант списать их с одного эмулятора и использовать повсеместно не подходит. Таким образом появилась необходимость запрашивать у QEMU какие устройства доступны в конкретной текущей версии.

Для частичного решения этой задачи подходит QMP. Наибольший интерес вызывают так называемые `query-commands`, команды, представляющие информацию по конкретным запросам.

QMP предоставляет достаточно большое количество команд для интроспекции конфигурации эмулятора. Мы используем следующие команды:

- `query-machines` – список машин;
- `query-cpu-definitions` – список процессоров;

- `qom-list-types` – информация о всех устройствах эмулятора. Предоставляется в виде списка соответствий «устройство» – «родитель»;
- `device-list-properties` – подробная информация о конкретном устройстве, заданном параметром.

Ниже приведены примеры команд и вывода.

```
-> {"execute": "query-machines"}
<- {"return": [{"hotpluggable-cpus": true, "name": "pc-i440fx-2.2", "cpu-max": 255}, {"hotpluggable-cpus": true, "name": "pc-q35-2.4", "cpu-max": 255}, ... {"hotpluggable-cpus": true, "name": "pc-i440fx-2.8", "cpu-max": 255}, ... {"hotpluggable-cpus": true, "name": "pc-1.0", "cpu-max": 255}]}
-> "execute": "device-list-properties", "arguments": {"typename": "e1000"}}
<- {"return": [{"name": "bootindex", "type": "int32"}, {"name": "mitigation", "description": "on/off", "type": "bool"}, {"name": "addr", "description": "Slot and optional function number, example: 06.0 or 06", "type": "int32"}, ... {"name": "vlan", "description": "Integer VLAN id to connect to", "type": "int32"}, {"name": "romfile", "type": "str"}, {"name": "rombar", "type": "uint32"}, {"name": "autonegotiation", "description": "on/off", "type": "bool"}, {"name": "netdev", "description": "ID of a netdev to use as a backend", "type": "str"}]}
```

Используя команду `qom-list-types` можно построить дерево всех устройств, однако для полноты картины не будет хватать шин. Вывод этой команды содержит только названия устройств. Характеристики каждого устройства можно запрашивать с помощью команды `device-list-properties`. Однако, в дереве будут отображены все возможные устройства, доступные для этой машины, а не только относящиеся к текущей конфигурации, то же относится и к характеристикам устройств.

Команды, аналогичной команде монитора `info qtree`, предоставляющей список устройств, которые установлены в машине в данный момент, в QMP нет, и именно поэтому решение частичное.

## *5. Реализация*

В рамках этой работы разрабатывался графический интерфейс для QEMU. Все вышеописанные манипуляции нашли применение в этом приложении.

Графические интерфейсы для эмулятора QEMU разрабатывались и ранее, например, AQEMU [6], QtEmu [7], JavaQemu[8]. Однако последние обновления этих инструментов зафиксированы несколько лет назад, они не поддерживают работу с детерминированным воспроизведением, а также не предоставляют возможности загрузить несколько версий эмулятора для удобного использования разных сборок. В нашем случае детерминированное воспроизведение является важным аспектом, поэтому было принято решение разрабатывать собственную графическую оболочку.

### 5.1 Управление состоянием эмулятора

Эмулятор может находиться в трех состояниях: работающий, остановленный, выключенный. Управление состоянием реализуется с помощью команд QMP: `cont` (возобновляет работу эмулятора после остановки), `stop` (пауза), `shutdown` (завершение работы эмулятора).

Эмулятор QEMU может быть запущен с опцией `-S`, что означает, что он будет запущен остановленным. Так как команды действия посылаются эмулятору посредством кнопок графического интерфейса, то необходимо чтобы их состояние было корректным

(активна/неактивна). Поэтому при старте эмулятор получает команду запроса статуса (`query-status`), дающую информацию о его текущем состоянии.

## 5.2 Получение информации об устройствах

Получить полную структуру устройств и шин через QMP на данный момент нельзя, поэтому пока получены только списки машин, моделей процессоров и сетевых карт.

Данные об этих устройствах соответствуют конкретной сборке QEMU и извлекаются в момент добавления этой сборки в графический интерфейс. Данные собираются и распределяются по конфигурационным файлам.

Для списка машин и моделей процессоров есть соответствующие команды `query-machines` и `query-cpu-definitions` соответственно. Списки машин и моделей процессоров обширны и неопытный пользователь может растеряться. Команда запроса информации о машинах имеет необязательный параметр `is-default`, который дает возможность узнать, какая машина использовалась бы QEMU по умолчанию. К сожалению, для процессора такого параметра нет, поскольку процессор по умолчанию зависит от машины. Поэтому в список процессоров добавлен вариант `default`, который отключит опцию командной строки, отвечающую за процессор, и QEMU назначит его по умолчанию самостоятельно.

Информация о других устройствах не может быть получена за один запрос. Сначала необходимо получить список всех устройств, затем запросить информацию по каждому из них и, исходя из результата, собирать нужные данные. Список устройств получаем командой `qom-list-types`. Далее команда `device-list-properties` должна отработать с каждым устройством из этого списка. Так как мы собирали информацию о сетевых картах, то в свойствах устройств искали указывающий на это параметр, а именно тип устройства `netdev`.

На данный момент структура дерева поддерживаемых устройств задана жестко. Машина создается с определенным минимальным набором устройств и шин (CPU, Memory, PCI, IDE). Эти устройства и шины не могут быть удалены. Некоторые устройства считаются неизменяемыми (например, машина), остальные могут быть отредактированы. Собственно, нередактируемые устройства в дереве не отображаются. Данные для редактирования некоторых устройств берутся из конфигурационных файлов, полученных с помощью QMP (модели ЦП, сетевые карты).

Добавляются устройства непосредственно на шину. Какие именно устройства могут быть добавлены, зависит от типа шины. Например, на шину PCI на данный момент можно добавить контроллеры IDE, SCSI и Network.

## 5.3 Определение списка снимков для запуска детерминированного воспроизведения

Снимки системы сохраняются в оверлей-файле. Этот файл является надстройкой над образом диска, в котором сохраняются все изменения, которые пользователь произвел с диском. Использование оверлея позволяет файлу образа диска оставаться неизменным. Чтобы пользователь мог указывать с какого снимка он хочет начать воспроизведение, мы должны предоставить ему список снимков, содержащихся в оверлее.

Детерминированное воспроизведение было разработано в ИСП РАН, и, на данный момент, в основную ветку QEMU приняты не все патчи. В связи с этим некоторую функциональность нельзя использовать в QemuGUI, рассчитанным на все версии QEMU (начиная с 2.8).

В данном случае речь идет о команде QMP `info_snapshots`, реализованной в ИСП РАН.

```
 -> { "execute": "info_snapshots" }
 <- { "return": [
          {
             "id": "1",
             "name": "retrace_0",
             "date": "2017-03-31",
             "time": "16:47:22",
             "vm-time": "00:00:00.000",
             "icount": 123
          },
          {
             ...
          }
       ]}
```

Вывод этой команды дает исчерпывающую информацию о снимках, но, к сожалению, с любыми версиями эмулятора воспользоваться ей пока что нельзя.

Другим путем получения этой информации является использование утилиты `qemu-img`, поставляемой вместе с эмулятором. Плюсом такого подхода является возможность получения списка снимков без запуска эмулятора (что не получилось бы при использовании команды QMP), а минусом вывод, разбор которого основан исключительно на том, как он выглядит. В случае изменения формата вывода разработчиками QEMU, мы столкнемся с необходимостью его переделывать. Сами разработчики эмулятора рекомендуют всегда использовать QMP.

Пример запроса и вывода:

```
->     qemu-img info C:/overlay0.ovl
<-     image: C:/overlay0.ovl
file format: qcow2
virtual size: 2.0G (2145386496 bytes)
disk size: 2.1M
cluster_size: 65536
backing file: C:/VBoxFolder/BSD.img
Snapshot list:
ID   TAG    VM SIZE   DATE                 VM CLOCK      ICOUNT
1    init   1.5M      2019-09-30 18:57:35  00:00:00.000  0
2    init   0         2019-09-30 18:57:35  00:00:00.000  0
Format specific information:
        compat: 1.1
        lazy refcounts: false
        refcount bits: 16
        corrupt: false
```

Следовательно, из всей этой информации нам нужен только раздел `Snapshot list`.

## 5.4 Описание графического интерфейса

Был реализован графический интерфейс [9], поддерживающий детерминированное воспроизведение. Главное окно программы представлено на рис. 2. В интерфейс можно добавлять неограниченное количество сборок QEMU, причем предусмотрена проверка совместимости исполняемого файла эмулятора и машины. Если машина не совместима с экземпляром эмулятора, то запустить его не удастся ни в обычном режиме, ни в режиме детерминированного воспроизведения.

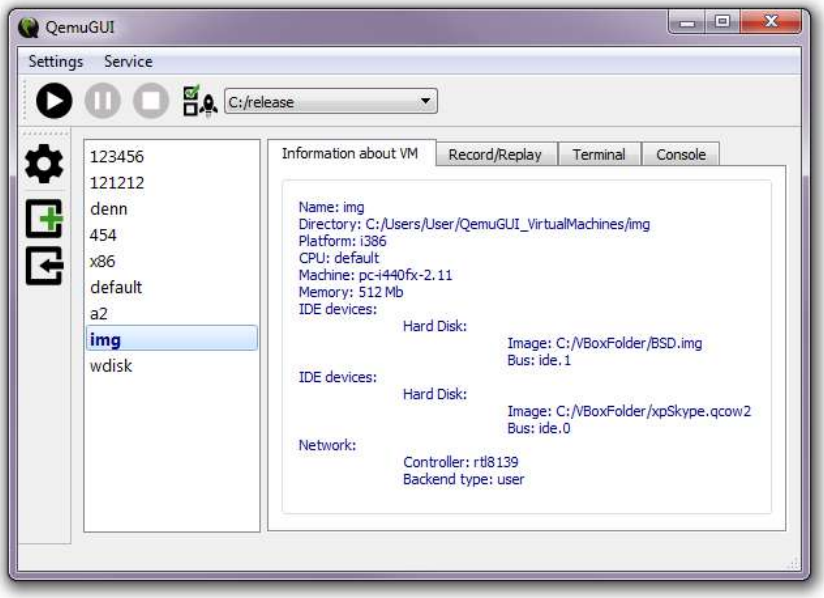


*Рис. 2. Главное окно программы QemuGUI*
*Fig. 2. QemuGUI main window*

QemuGUI позволяет создавать машины и конфигурировать их, окно создания машины представлено на рис. 3, а конфигурирования на рис. 4. В один момент времени может быть запущен только один экземпляр QEMU.

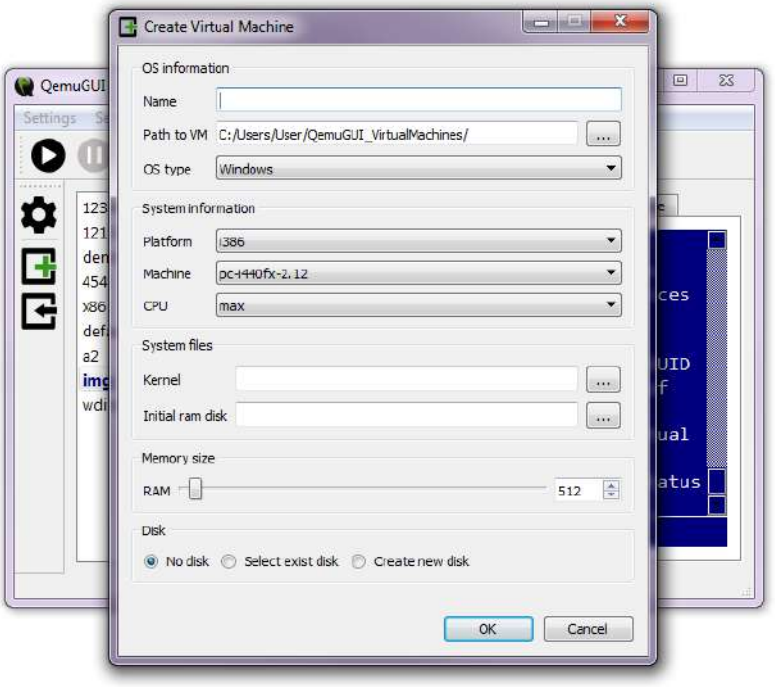


*Рис. 3. Форма создания новой машины*
*Fig. 3. Create machine wizard*

Так как на данный момент поддерживается достаточно узкий круг устройств, есть возможность написать дополнительные опции командной строки. Такие поля есть в форме редактирования устройства (в таком случае опции сохранятся) и в окне `Run options` (изменения этого окна не сохраняются). Помимо дополнительной командной строки в этом окне можно включить лог-файлы QEMU, а также установить флаги: запускать остановленной, запускать в режиме отладки и запускать с опцией `snapshot`. Окно представлено на рис. 5.

Основная область главного окна QemuGUI содержит четыре вкладки: информация о машине, детерминированное воспроизведение, монитор и консоль для вывода информации от эмулятора. На данный момент туда выводится командная строка при запуске QEMU.

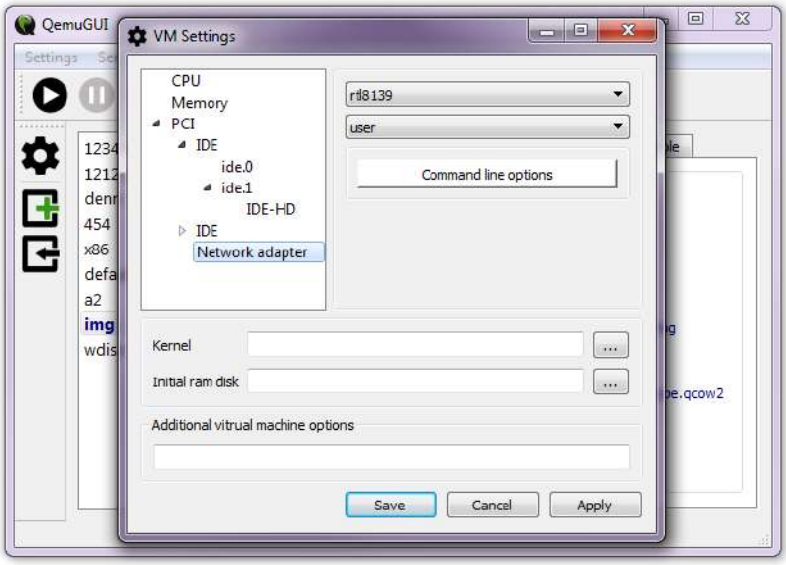


*Рис. 4. Дерево устройств и их настройка*
*Fig. 4. Tree of devices and their settings*

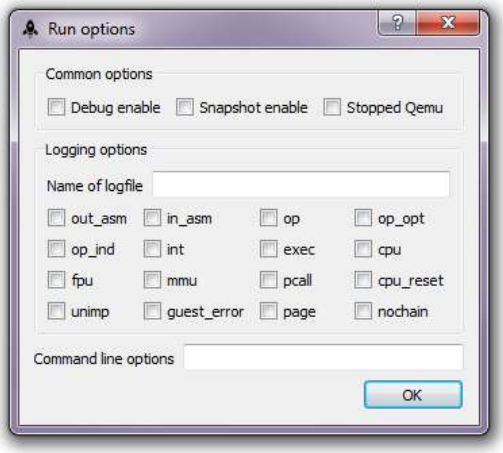


*Рис.5. Окно настроек для запуска QEMU*
*Fig. 5. QEMU launch settings window*

На вкладке детерминированного воспроизведения находится список текущих записанных сценариев, а также непосредственно кнопки для записи и воспроизведения, представлено на рис. 6. Если сценарий записывается с опцией создания снимка, то при воспроизведении будет предложен выбор снимка, с которого будет начато воспроизведение. По умолчанию создается снимок `init`, если была включена опция автоматического создания снимков, они будут называться `auto_N` и будут представлены в выпадающем списке.

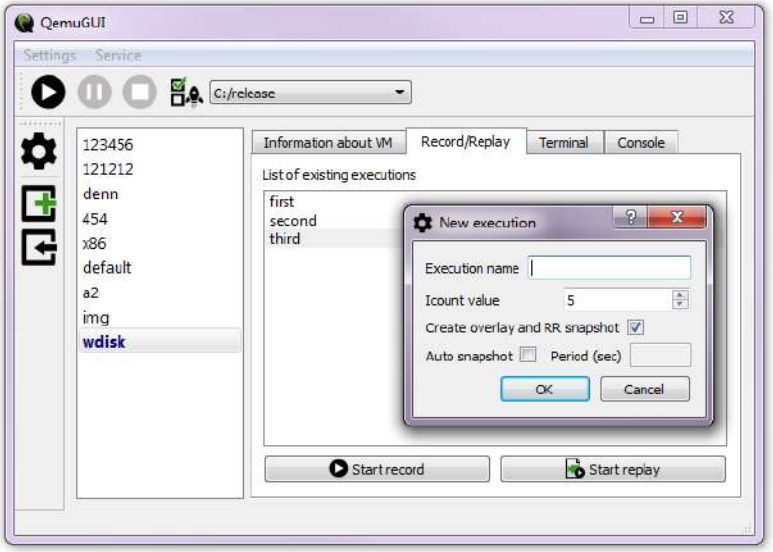


*Рис. 6. Создание нового журнала*
*Fig. 6. Create execution wizard*

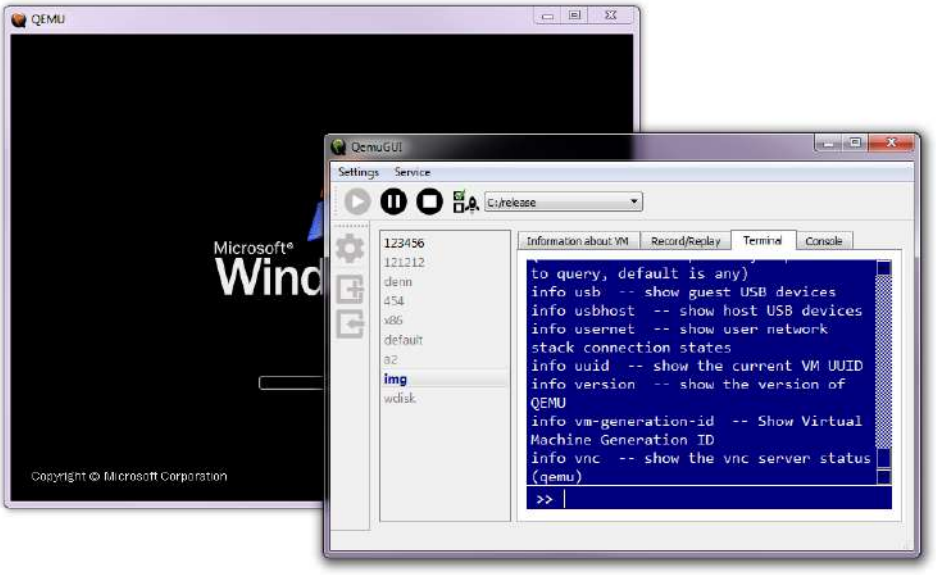


*Рис. 7. Монитор QEMU*
*Fig. 7. QEMU monitor*

Вкладка с монитором представлена на рис. 7. Наличие монитора в интерфейсе делает использование эмулятора более удобным. Внешний вид терминала можно менять в настройках. Также в настройках можно менять порты для подключения QMP и монитора.

Все данные и настройки, используемые QemuGUI, хранятся в конфигурационных файлах в формате XML.

К недостаткам QemuGUI можно отнести ограниченный набор виртуальных устройств, поддерживаемых для добавления в дерево. Для смягчения этого недостатка в интерфейсе присутствует поле для дополнительных опций эмулятора, с помощью которого можно вручную дополнить конфигурацию машины.

Графический интерфейс находится в открытом доступе [10].

## *6. Заключение*

В результате был разработан графический интерфейс для эмулятора QEMU, позволяющий получать актуальные сведения о некоторых устройствах, а также помогающий пользователям более удобно и эффективно работать с QEMU, в частности, и с детерминированным воспроизведением. Графическая оболочка работает под операционными системами Windows и Linux.

## References

[1]. QEMU the FAST! processor emulator. Available at: https://www.qemu.org (accessed 07.11.2019)
[2]. QEMU code. Available at: https://git.qemu.org/git/qemu.git (accessed 12.10.2019)
[3]. QEMU interface introspection: From hacks to solutions. Markus Armbruster. KVM Forum 2015. Available at: https://events.static.linuxfound.org/sites/events/files/slides/armbru-qemu-introspection.pdf (accessed 07.11.2019)
[4]. QEMU User Documentation. Available at: https://qemu.weilnetz.de/doc/qemu-doc.html (accessed 12.10.2019)
[5]. Pavel Dovgalyuk. Deterministic Replay of System's Execution with Multi-target QEMU Simulator for Dynamic Analysis and Reverse Debugging. In Proc. of 16th European Conference on Software Maintenance and Reengineering, 2012, vol. 1, pp. 553-556.
[6]. QEMU Monitor documentation. Available at: https://en.wikibooks.org/wiki/QEMU/Monitor (accessed 07.11.2019)
[7]. QEMU Monitor. Available at: http://people.redhat.com/pbonzini/qemu-test-doc/_build/html/topics/pcsys_005f monitor.html (accessed 07.11.2019)
[8]. QMP Documentation. Available at: https://wiki.qemu.org/Documentation/QMP (accessed я 07.11.2019)
[9]. Introducing JSON. Available at: http://www.json.org/ (accessed 07.11.2019)
[10]. AQEMU. Available at: https://sourceforge.net/projects/aqemu/ (accessed 07.11.2019)
[11]. QtEmu. Available at: https://qtemu.org/ (дата обращения 07.11.2019)
[12]. JavaQemu. Available at: https://sourceforge.net/projects/javaqemu/ (accessed 07.11.2019)
[13]. Qt Documentation. Available at: https://doc.qt.io/qt-5/qt5-intro.html (accessed 07.11.2019)
[14]. QemuGUI source. Available at: https://github.com/ispras/qemu-gui (accessed 07.11.2019)

## Информация об авторах / Information about authors

Наталья Игоревна ФУРСОВА – старший научный сотрудник, старший преподаватель, кандидат технических наук. Сфера научных интересов: интроспекция и инструментирование виртуальных машин, динамический анализ кода, эмуляторы.

Natalia Igorevna FURSOVA – senior researcher, senior lecturer, Ph.D in Technical Sciences. Research interests: virtual machines introspection and instrumentation, dynamic analysis of code, emulators.

Павел Михайлович ДОВГАЛЮК – старший научный сотрудник, доцент, кандидат технических наук. Сфера научных интересов: интроспекция и инструментирование виртуальных машин, динамический анализ кода, эмуляторы.

Pavel Michailovich DOVGALYUK – senior researcher, associate professor, Ph.D. in Technical Sciences. Research interests: virtual machines introspection and instrumentation, dynamic analysis of code, emulators.

**DOI: 10.15514/ISPRAS-2019-31(5)-3**



# Автоматическое доказательство корректности программ с динамической памятью

[1] *Ю.О.Костюков, ORCID: 0000-0003-4607-039X <kostyukov.yurii@gmail.com>*
[1] *К.А.Батоев, ORCID: 0000-0003-1124-7909 <konstantin.batoev@gmail.com>*
[1,2] *Д.А.Мордвинов, ORCID: 0000-0002-6437-3020 <dmitry.mordvinov@jetbrains.com>*
[1] *М.П.Костицын, ORCID: 0000-0001-9982-6571 <mishakosticyn@yandex.ru>*
[1] *А.В.Мисонижник, ORCID: 0000-0002-5907-0324 <misonijnik@gmail.com>*

[1] *Санкт-Петербургский государственный университет,*
*199034, Россия, Санкт-Петербург, Университетская набережная, д. 7–9*
[2] *JetBrains Research*
*197342, Россия, Санкт-Петербург, Кантемировская ул., 2*

**Аннотация**. В данной работе изучаются теоретические основы автоматической модульной верификации императивных программ с динамической памятью. Вводится формализм *композициональной символьной памяти*, который используется для построения композиционального алгоритма, порождающего *обобщённые кучи*. Они являются термами *исчисления символьных куч*, которые описывают состояния *произвольных* циклических фрагментов программы. Выводимые в этом исчислении *кучи* соответствуют достижимым состояниям исходной программы. В работе также устанавливается соответствие между выводом в этом исчислении и исполнением функциональных программ второго порядка без эффектов.

**Ключевые слова:** формальная верификация; автоматическая верификация; символьное исполнение; анализ программ; динамическая память; композициональность; чистые функции

**Для цитирования:** Костюков Ю.О., Батоев К.А., Мордвинов Д.А., Костицын М.П., Мисонижник А.В. Автоматическое доказательство корректности программ с динамической памятью. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 37-62. DOI: 10.15514/ISPRAS-2019-31(5)-3

## Automatic verification of heap-manipulating programs

[1] *Yu.O. Kostyukov, ORCID: 0000-0003-4607-039X <kostyukov.yurii@gmail.com>*
[1] *K.A. Batoev, ORCID: 0000-0003-1124-7909 <konstantin.batoev@gmail.com>*
[1,2] *D.A. Mordvinov, ORCID: 0000-0002-6437-3020 <dmitry.mordvinov@jetbrains.com>*
[1] *M.P. Kostitsyn, ORCID: 0000-0001-9982-6571 <mishakosticyn@yandex.ru>*
[1] *A.V. Misonizhnik, ORCID: 0000-0002-5907-0324 <misonijnik@gmail.com>*

[1] *Saint Petersburg State University,*
*3A bld. 1, Kantemirovskaya st., St Petersburg, Russia, 194100*
[2] *JetBrains Research*
*2, Kantemirovskaya st., St Petersburg, Russia, 197342*

**Abstract.** Theoretical foundations of compositional reasoning about heaps in imperative programming languages are investigated. We introduce a novel concept of compositional symbolic memory and its relevant properties. We utilize these formal foundations to build up a compositional algorithm that generates

generalized heaps, terms of symbolic heap calculus, which characterize arbitrary cyclic code segments. All states inferred by this calculus precisely correspond to reachable states of the original program. We establish the correspondence between inference in this calculus and execution of pure second-order functional programs. The contribution of this work is as follows: (1) a formal model of compositional symbolic memory is proposed; (2) the properties of its correctness are formulated; (3) the calculus of symbolic heaps has been introduced: the conclusions in this calculus give all attainable states of the program; (4) the concept of generalized heaps is introduced, an algorithm for automatic modular construction of generalized heaps according to an imperative program is proposed; (5) an approach is proposed to reduce the problem of finding an output in calculus of symbolic heaps to the problem of proving the safety of functional programs.

**Keywords:** formal verification; automatic verification; symbolic execution; static analysis; dynamic memory; heap analysis; compositionality; pure functions

**For citation:** Kostyukov Yu.O., Batoev K.A., Mordvinov D.A., Kostitsyn M.P., Misonizhnik A.V. Automatic verification of heap-manipulating programs. Trudy ISP RAN/Proc. ISP RAS, vol.31, issue 5, 2019, pp. 37-62 (in Russian). DOI: 10.15514/ISPRAS-2019-31(5)-3

## *1. Введение*

Б*о*льшая часть современного программного обеспечения написана на языках с динамически выделяемой памятью, таких как C++, Java, C#. Автоматическая верификация и анализ таких программ является очень трудоёмкой задачей [1]. При этом даже корректные с теоретической точки зрения методы могут оказаться неэффективными из-за больших размеров анализируемых программ [2]. Для решения этой проблемы были предложены *композициональные* техники, которые хорошо зарекомендовали себя на практике [3, 4, 5, 6]. Такие техники выполняют анализ функций в *изоляции*, т. е. вне контекста конкретного вызова, и в дальнейшем переиспользуют промежуточные результаты анализа. Таким образом, можно свести верификацию больших систем к задаче верификации набора небольших фрагментов кода.

Большинство существующих композициональных техник являются *неточными* в том смысле, что они аппроксимируют пространство состояний программы снизу или сверху. Аппроксимирующие снизу подходы рассматривают не все сценарии поведения программы, например, при помощи раскрутки циклов на конечное число шагов [7]. Это позволяет находить ошибки, но не доказывать корректность произвольных программ. Аппроксимирующие сверху подходы анализируют упрощённую версию программы, что на практике приводит к большому числу ложноположительных срабатываний [8]. Таким образом, остаётся актуальной задача точного анализа программ с помощью композициональных техник.

Эту задачу можно решить при помощи введения особой *модели памяти*, представляющей состояния программы. Такая модель должна быть достаточно гибкой и выразительной, чтобы с её помощью можно было описать *произвольные* свойства программы, тем самым охватив все сценарии её поведения. Свойства программ в модели могут быть выражены в виде логических формул.

В данной статье предложен подход к модульной верификации программ с динамической памятью. Подход основывается на новой модели *композициональной символьной памяти*. Эта модель основывается на идее *ленивого инстанцирования* [9] и описывает состояние динамической памяти программы символьными выражениями над значениями входных *ячеек памяти*. Адреса этих ячеек могут быть символьно описаны с использованием содержимого других ячеек; это позволяет выражать эффект любой функции на произвольной рекурсивной структуре данных.

Модель композициональной символьной памяти используется для построения *исчисления символьных куч*, которые, в свою очередь, позволяют описывать поведение произвольной императивной программы с динамической памятью при помощи символьных состояний

программы. Термы этого исчисления (т.н. *обобщённые кучи*), построенные по программе, *в точности* описывают эффект этой программы на произвольном состоянии.

В статье также предложена автоматическая процедура композиционального построения обобщённых куч, описывающих произвольные фрагменты императивной программы.

Поскольку для вывода в исчислении символьных куч не требуется хранения контекста, по любой обобщённой символьной куче можно построить эквивалентную ей функциональную программу, сведя задачу проверки корректности императивных программ с динамической памятью к задаче проверки корректности чистых функций. В статье предлагается сведение задачи анализа императивных программ к задаче вывода *уточнённых типов* (refinement types) [10] для функциональных программ, получаемых из обобщённых куч.

Доказательства сформулированных в работе свойств и теорем в основном опущены в виду ограничений на размер статьи. Читатель может ознакомиться с ними в [27].

Вклад данной работы заключается в следующем: (1) предложена формальная модель *композициональной символьной памяти*; (2) сформулированы свойства её корректности; (3) введено *исчисление символьных куч*: выводы в этом исчислении дают все достижимые состояния программы; (4) введено понятие *обобщённых куч*, предложен алгоритм автоматического модульного построения обобщённых куч по императивной программе; (5) предложен подход к сведению задачи поиска вывода в *исчислении символьных куч* к задаче доказательства безопасности функциональных программ.

## *2. Обзор*

Существует большое количество работ, посвящённых автоматическому анализу и доказательству корректности программ с динамической памятью [1, 2, 5, 6]. Подходы делятся на *аппроксимирующие снизу*, *аппроксимирующие сверху* и *точные*. Аппроксимирующие снизу подходы исследуют только часть пространства состояний; они пригодны для поиска ошибок в программах, но не годятся для доказательства корректности программ. Подходы, аппроксимирующие пространство состояний сверху, пригодны для доказательства корректности программ, но на практике порождают большое количество ложноположительных срабатываний. Точные подходы исследуют все пространство состояний, но, насколько известно авторам, в данной статье представлен первый *полностью автоматический* подход к точному анализу императивных программ с динамической памятью.

К подходам, аппроксимирующим пространство состояний снизу, можно отнести динамическое символьное исполнение [11] и ограничиваемую проверку моделей [12]. Одна из основных идей в символьном исполнении программ с динамической памятью – ленивое инстанцирование [9]. Оно позволяет производить анализ программ с рекурсивными структурами данных без ручной спецификации размеров этих структур. Существует множество работ, развивающих эту идею [13, 14], но все они следуют идее классического символьного исполнения с раскруткой отношения перехода. Алгоритм, представленный в данной работе, не раскручивает отношение перехода программы, а строит систему ограничений в формализме композициональной символьной памяти, точно описывающих поведения программы.

Как правило, подходы, аппроксимирующие пространство состояний сверху, основаны на *абстрактной интерпретации* программ [2, 6, 8]. Существует множество подходов к абстрактной интерпретации программ с динамической памятью. Подавляющее большинство из них основаны на *логике с разделением* (separation logic) [15]; абстрактный домен таких анализаторов хорошо подходит для автоматического анализа *формы куч* [16], т.е. анализом того, как объекты в куче *связаны* друг с другом, а не их содержимого.

Анализаторы, которые пользуются логикой с разделением, являются, как правило, композициональными [2, 5, 6]. Логика с разделением адаптирована для символьного исполнения программ [17]. Основной проблемой логики с разделением является её невыразительность — она хорошо подходит для рассуждения о форме куч, но не подходит для анализа *данных* в динамической памяти, потому что введённая в этой логике разделяющая конъюнкция не позволяет выражать сложных свойств. Существуют подходы на основе логики с разделением, которые позволяют производить более точный анализ динамической памяти вплоть до побайтовых манипуляций с памятью [18]. Однако практическая применимость таких подходов сопряжена с большим количеством ложноположительных срабатываний.

Существуют также работы, основанные на идее сведения задачи верификации программ с динамической памятью к решению системы рекурсивно-логических ограничений [3, 20]. Такие подходы, как и наш, используют SMT-решатели [19] для решения ограничений и вывода индуктивных инвариантов системы. Логические решатели позволяют верифицировать программы, не порождая ложных срабатываний.

## 3. Демонстрационный язык

В данном разделе мы определяем демонстрационный язык программирования с операциями над динамической памятью, для которого позже будет представлена процедура автоматической модульной верификации. Синтаксис языка представлен на рис. 1.

$$
\begin{array}{l}
Program ::= Statement^* \\
Statement ::= label : Statement \\
\quad\quad | \; Location := Expression \\
\quad\quad | \; \boldsymbol{goto}\ \{Expression \rightarrow label\}^+ \\
\quad\quad | \; \boldsymbol{fail} \mid \boldsymbol{halt} \\
Expression ::= \boldsymbol{null} \mid \boldsymbol{true} \mid \boldsymbol{false} \mid \mathbb{N} \\
\quad\quad | \; Expression\ BinOp\ Expression \\
\quad\quad | \; UnOp\ Expression \\
\quad\quad | \; Location \\
\quad\quad | \; \boldsymbol{new}\ \{ident\ =\ Expression\}^+ \\
Location ::= ident \mid Location.ident
\end{array}
$$

*Рис. 1. Синтаксис демо-языка*
*Fig. 1. Demo language syntax*

Состояния программы на этом языке описываются состояниями динамической памяти. Язык не содержит функций, операций ветвления и циклов, однако включает оператор условного перехода по метке $\boldsymbol{goto}\ \{Expression \rightarrow label\}^+$. Слева от стрелки находится условие перехода, справа — метка, на которую нужно перейти. Оператор последовательно вычисляет условия слева от стрелок и переходит по первой метке справа от стрелки, чьё условие истинно.

Идентификаторами $ident$ в языке называются идентификаторы в стиле языка C; $BinOp$ *и* $UnOp$ это простые арифметические и булевы операторы.

Оператор $\boldsymbol{new}\ \{ident\ =\ Expression\}^+$ выделяет в памяти новый объект с именами полей $ident$, инициализируя каждый из них соответствующим $Expression$. Используя точку, как в определении $Location$, можно получить доступ к полям объектов. Доступ может быть вложенный, например $list.RootNode.Next.Key$.

В каждой переменной находятся либо примитивные значения (**null**, **true**, 42), либо ссылки на объекты в динамической памяти. Оператор $:=$ перезаписывает ссылку. Например, в листинге 1 в строке 3 в переменную $p$ помещается ссылка на $l$, сам объект не копируется. Затем, в строке 7 переменная $p$ начинает ссылаться на новый объект, а переменная $l$ по-

прежнему указывает на старый. В строке 10 меняется само содержимое памяти, независимо от переменных *p и l*.

```
1.  RemoveAll:
2.  l := new {Key = x, Next = l}
3.  p := l
4.  RemoveAllIterate:
5.  goto {p.Next = null  -> RemoveAllFinalize,
6.        p.Next.Key = x -> RemoveAllRemoveElement}
7.  p := p.Next
8.  goto {true -> RemoveAllIterate}
9.  RemoveAllRemoveElement:
10. p.Next := p.Next.Next
11. goto {true -> RemoveAllIterate}
12. RemoveAllFinalize:
13. l := l.Next
14.
15. p := l
16. Contains:
17. goto {p = null  -> Exit,
18.       p.Key = x -> Error}
19. p := p.Next
20. goto {true -> Contains}
21.
22. Error: fail
23. Exit: halt
```

*Листинг 1. Программа, модифицирующая динамическую память*
*Listing 1. Example of heap-manipulating program.*

Далее мы будем рассматривать только корректно типизированные программы: арифметические операции применяются согласно их стандартной семантике; поле и то, что в него записывается, согласованы по типам; выражения в **goto** имеют булевский тип; поля всегда читаются успешно (кроме чтения из **null**); выражения и имена полей, используемые в операторе **new**, согласованы по типам. Также мы предполагаем, что программы не читают из неинициализированных локаций, всякая программа содержит инструкцию **halt**, все метки, на которые есть переходы, определены и имена идентификаторов, меток и ключевые слова языка не пересекаются. Предложенный язык не даёт возможности управлять памятью на низком уровне, в частности, освобождать выделенную память. Таким образом, программы на этом языке допускают всего два вида ошибок, которые необходимо находить: доступ к полю по ссылке, содержащей **null**, и достижимость инструкции **fail**.

## *4. Композициональная символьная память*

В центре нашего подхода стоит формализм композициональных символьных куч. Концепция *композициональной символьной памяти* (КСП), определяемая в этом разделе, основана на идее *ленивого инстанцирования* [9, 13]. Ленивое инстанцирование — это техника, позволяющая строить *конечные* символьные выражения для рекурсивных структур данных, таких как связные списки и деревья. При этом предлагается инициализировать поля рекурсивных структур данных *по требованию* вместо инициализации всей структуры одномоментно, что потребовало бы заранее заданных ограничений на её размер. Это, например, позволяет анализировать списки и деревья, не зная заранее их размер.

| | |
|---|---|
| 1. `goto {true -> F}` | C |
| 2. `G:` | |
| 3. `x := x + 1` | G (стр. 2-4): $\{x \mapsto LI(x) + 1\}$ |
| 4. `goto {true -> Exit}` | F (стр. 5-7): $\{x \mapsto 42\} \circ \{x \mapsto LI(x) + 1\} =$ |
| 5. `F:` | $= \{x \mapsto 42 + 1\}$ |
| 6. `x := 42` | |
| 5. `goto {true -> G}` | |
| 8. `Exit: halt` | |

*Рис. 2. Пример композиции состояний*
*Fig. 2. Heap composition example.*

Основная идея КСП состоит в том, чтобы трактовать лениво инициализированные локации $LI(x)$[1] как *ячейки*, в которые будут подставлены значения из контекстного состояния. Например, состояние, описывающее код после метки *G* на рис. 2 (стр. 2-4), содержит незаполненную ячейку *x*. Это означает, что оно описывает *эффект* этого фрагмента кода на *произвольном* контекстном состоянии, т.е. состоянии с произвольным значением *x*. Таким контекстным состоянием может быть, например, состояние после метки *F* до перехода на *G* (стр. 5-6). Чтобы получить полное состояние после метки *F* (стр. 5-7), необходимо заранее подсчитанный эффект *G* применить к текущему состоянию *F*. Для этого нужно заполнить ячейки состояния *G* значениями из текущего контекста *F*. Кучу, полученную в результате этого процесса, мы называем *композицией* куч. Заметим, что адаптация этой идеи к программам произвольной сложности требует некоторых дополнительных усилий.

Далее будет описан формализм композициональной символьной памяти. Доказательства теорем приведены в [27].

## 4.1 Символьные выражения

Чтобы представлять произвольные состояния программ на нашем демо-языке, необходимо ввести понятие символьных выражений. Определение символьного выражения представлено на рис. 3.

Символьный терм ($term$) — это либо арифметическое выражение ($arith$), либо символьный адрес локации в памяти ($loc$).

$$
\begin{aligned}
term &::= \ arith \mid loc \\
arith &::= \ \mathbb{N} \mid arith \pm arith \mid -arith \mid LI^{arith}(loc) \\
&\quad \mid union^{arith}(\langle guard, arith \rangle^*) \\
loc &::= \ null \mid 0x[0-9]^+ \mid ident \\
&\quad \mid loc.FieldName \mid LI^{loc}(loc) \\
&\quad \mid union^{loc}(\langle guard, loc \rangle^*) \\
guard &::= \ \top \mid \bot \mid \neg guard \mid guard \wedge guard \mid guard \vee guard \\
&\quad \mid arith = arith \mid arith < arith \mid loc = loc
\end{aligned}
$$

*Рис. 3. Грамматика символьных выражений*
*Fig. 3. Symbolic terms.*

[1] LI — lazy instantiation

Символьные значения записываются как $LI^*(x)$, что означает «ленивое инстанцирование $x$». Далее всюду тип символьного значения либо не важен, либо очевиден из контекста, потому мы будем его опускать и писать просто $LI(x)$. Заметим, что локация-источник символьного значения $LI$ может быть также символьной, так что, например, термы вида $LI(LI(list).Key) + 1$ также допустимы.

```
1. a := new {Key = 15, Next = null}
2. a.Next := a
3. Next := a
4. c := d.Next
```

$$\begin{cases} a & \mapsto 0x1 \\ 0x1.Key & \mapsto 15 \\ 0x1.Next & \mapsto 0x1 \\ LI(b).Next & \mapsto 0x1 \\ c & \mapsto LI(LI(d).Next) \end{cases}$$

*Рис. 4. Пример различных типов локаций*
*Fig. 4. Different location types example.*

Локации могут быть *конкретными*, т.е. известными в текущем контексте ($0x[0-9]^+$, порождаются оператором **new**); именованными ($ident$, глобальные переменные, $a, b, c$); ссылками на конкретное поле; ленивыми инстанцированиями других локаций. Например, на рис. 4, где представлен фрагмент кода и соответствующее ему символьное состояние памяти, локация $b$ неизвестна, поэтому запись в её поле $Next$ порождает запись по символьной локации $LI(b).Next$. Аналогично неизвестна локация $d$, но также неизвестен и элемент, на который ссылается её поле $Next$, из-за чего символьная локация $c$ указывает на символьный терм $LI(LI(d).Next)$.

Вернёмся к определению символьных выражений (рис. 3). На этом рисунке $guard$ обозначает *ограничение,* представленное в виде логической формулы (условие пути или условие перехода по метке). Такие формулы «защищают» элементы *символьных объединений*. Символьное объединение[2] ($union$) — это обобщение символа $ite(cond, x, y)$. Как и в случае символьных значений, мы опускаем тип символьных объединений и пишем просто $union(*)$. За символьным объединением мы закрепляем следующую семантику: $x = union(\langle g_1, v_1\rangle, \dots, \langle g_n, v_n\rangle)$ тогда и только тогда, когда $(g_1 \wedge x = v_1) \vee \dots \vee (g_n \wedge x = v_n)$. Символьные объединения позволяют при помощи одного символьного состояния описать несколько веток исполнения программы.

**Замечание.** Потребуем, чтобы ограничения в объединениях *не пересекались*, т. е. только одно ограничение должно выполняться при подстановке конкретных значений вместо символьных. Однако несколько ограничений могут выполняться одновременно в том случае, если они «защищают» одно и то же значение. Например, допустимы объединения вида

$$union(\langle LI(x) = LI(y), LI(LI(y).Key) + 7\rangle,\\ \langle LI(x) = LI(z), LI(LI(z).Key) + 7\rangle).$$

Мы рассматриваем содержимое объединений как множество пар, потому пишем, например, $union(\{\langle g, v\rangle\} \cup X)$. Чтобы избежать перегрузки синтаксиса лишними скобками, мы опускаем их далее при записи одноэлементных множеств. Так, пример выше можно записать следующим образом: $union(\langle g, v\rangle \cup X)$. Также мы опускаем круглые скобки там, где не возникает двусмысленности, например, пишем $union\langle x > 5{,}42\rangle$ или $union\{\langle g_i, v_i\rangle | 1 \le i \le n\}$

*Выражение* на рис. 3 — это либо терм, либо ограничение. *Примитивные* выражения — это натуральные числа, именованные локации, конкретные адреса в памяти, $\boldsymbol{null}$, $\top$ и $\bot$. *Операциями* являются сложение, вычитание, унарный минус, сравнения, логические

[2] Понятие символьных объединений заимствовано из [21]

связки и чтение поля. В тех случаях, когда вид операции не важен, мы пользуемся следующей нотацией: $op(e_1, \dots, e_n)$.

**Замечание.** Равенство символьных термов является *семантическим*, например, $2 * (x + 1) = x + x + 4 - 2$ и $union(\langle x + 5 = y + 4{,}7\rangle, \langle \bot\ {,}42\rangle) = union\langle x + 1 = y, 7\rangle$.

Далее мы перечисляем некоторые очевидные свойства символьного объединения.

**Утверждение 1.**

(a) $union\langle \top, v\rangle = v$

(b) $union(\langle \bot, v\rangle \cup X) = union(X)$

$$union(\langle g, union(\langle g_1, v_1\rangle, \dots, \langle g_n, v_n\rangle)\rangle \cup X) = \\ = union(\{\langle g \wedge g_1, v_1\rangle, \dots, \langle g \wedge g_n, v_n\rangle\} \cup X)$$

В частности,

- $union(\langle g, union(\varnothing)\rangle \cup X) = union(X)$
- $union\langle g_1 \vee \dots \vee g_n, union(\langle g_1, v_1\rangle, \dots, \langle g_n, v_n\rangle)\rangle = \\ = union(\langle g_1, v_1\rangle, \dots, \langle g_n, v_n\rangle)$

(c) $union(\{\langle g_1, v\rangle, \dots, \langle g_n, v\rangle\} \cup X) = union(\langle g_1 \vee \dots \vee g_n, v\rangle \cup X)$

В частности,

$$union(\{\langle g, v\rangle, \langle g \wedge g_1, v\rangle, \dots, \langle g \wedge g_n, v\rangle\} \cup X) = union(\langle g, v\rangle \cup X)$$

(d) $op(union(\langle g_1^1, e_1^1\rangle, \dots, \langle g_{n_1}^1, e_{n_1}^1\rangle), \dots, union(\langle g_1^m, e_1^m\rangle, \dots, \langle g_{n_m}^m, e_{n_m}^m\rangle)) = \\ = union(\{\langle g_{i_1}^1 \wedge \dots \wedge g_{i_m}^m, op(e_{i_1}^1, \dots, e_{i_m}^m)\rangle | 1 \le i_j \le n_j\})$

В частности, для непересекающихся ограничений $g_1, \dots, g_n$

$$op(union(\langle g_1, e_1^1\rangle, \dots, \langle g_n, e_n^1\rangle), \dots, union(\langle g_1, e_1^m\rangle, \dots, \langle g_n, e_n^m\rangle)) = \\ = union(\langle g_1, op(e_1^1, \dots, e_1^m)\rangle, \dots, \langle g_n, op(e_n^1, \dots, e_n^m)\rangle)$$

## 4.2 Символьные кучи

**Определение 1.** *Символьная куча* — это частичная функция $\sigma: loc \to term$, удовлетворяющая следующему требованию *(инвариант кучи):*

$$\forall x, y \in dom(\sigma), union\langle x = y, \sigma(x)\rangle = union\langle x = y, \sigma(y)\rangle. \quad (1)$$

Заметим, что это более сильное ограничение, чем накладывает само понятие функции ($x \equiv y \Rightarrow \sigma(x) \equiv \sigma(y)$). Это связано с тем, что равенство термов и локаций в нашем подходе является не синтаксическим, а семантическим. Так, например, функция $\{LI(x).Key \mapsto 10;\ LI(y).Key \mapsto 15\}$ не является символьной кучей, т.к. при подстановке вместо $x$ и $y$, например, $0x1$, получится, что этот адрес указывает на два разных значения. Напротив, следующая функция является символьной кучей: $\{LI(x).Key \mapsto union(\langle LI(x) = LI(y), 15\rangle, \langle LI(x) \neq LI(y), 10\rangle); LI(y).Key \mapsto 15\}$.

**Определение 2.** Пустая куча $\epsilon$ — это частичная функция с областью определения $dom(\epsilon) = \varnothing$ (она, очевидно, удовлетворяет (1)).

**Определение 3.** Пусть $x \in dom(\sigma)$ или $x$ — символьная локация. Тогда определим чтение локации $x$ в символьной куче $\sigma$ следующим образом:

$$read(\sigma, x) \stackrel{\text{def}}{=} union(\{\langle x = l, \sigma(l)\rangle | l \in dom(\sigma)\} \cup \langle \bigwedge_{l \in dom(\sigma)} x \neq l, LI(x)\rangle). \quad (2)$$

Интуитивно, чтение пытается сопоставить ссылку $x$ (возможно, символьную) с каждым адресом локации в $\sigma$ (также, возможно символьным). Если ссылка и некоторый адрес совпали, то результатом чтения будет значение, лежащее по этому адресу. Если не было найдено ни одного совпадения, то возвращается символьное значение $LI(x)$.

Очевидно, что при $x \in dom(\sigma)$ выполнено $read(\sigma, x) = \sigma(x)$. Одно из ограничений $x = l$ будет выполняться, тогда как один из элементов конъюнкции $\wedge_{l \in dom(\sigma)} x \neq l$ будет, наоборот, невыполним, следовательно $read(\sigma, x)$ не сможет вернуть значение $LI(x)$.

Сделаем следующее наблюдение: фактически, из опр. 3 следует, что множество ограничений в формуле (2) может содержать пересечения, т.е. в куче могут содержаться две (или более) символьные локации, которые совпадают при некоторых конкретных подстановках. Инвариант символьной кучи (1) позволяет обойти возможную проблему с совпадающими адресами: благодаря ему при совпадении ограничений не будет конфликтов между «защищаемыми» значениями.

**Пример 1.** *Пусть* $\sigma = \{0x1.A \mapsto 42; LI(x).B \mapsto union(\langle LI(x).B = 0x1.A, 42\rangle, \langle LI(x).B \neq 0x1.A, 7\rangle)\}$. *Тогда*

$$read(\sigma, 0x1.A) = union(\langle 0x1.A = 0x1.A, 42\rangle, \langle 0x1.A = LI(x).B, union(\dots)\rangle,$$
$$\langle 0x1.A \neq 0x1.A \wedge LI(x).B \neq 0x1.A, LI(0x1.A)\rangle) =$$
$$= union(\langle \top, 42\rangle, \langle 0x1.A = LI(x).B, union(\dots)\rangle, \langle \bot, LI(0x1.A)\rangle) = 42$$
$$read(\sigma, LI(y).B) =$$
$$= union(\langle LI(y).B = 0x1.A, 42\rangle, \langle LI(y).B = LI(x).B,$$
$$union(\langle LI(x).B = 0x1.A, 42\rangle, \langle LI(x).B \neq 0x1.A, 7\rangle)\rangle,$$
$$\langle LI(y).B \neq 0x1.A \wedge LI(y).B \neq LI(x).B, LI(LI(y).B)\rangle) \overset{\text{утв. 1}}{=}$$
$$= union(\langle LI(y).B = 0x1.A, 42\rangle, \langle LI(y).B = LI(x).B \wedge LI(x).B \neq 0x1.A, 7\rangle,$$
$$\langle LI(y).B \neq 0x1.A \wedge LI(y).B \neq LI(x).B, LI(LI(y).B)\rangle)$$

## 4.3 Композиция символьных куч

**Определение 4.** *Уточнение* выражения $e$ в контексте символьной кучи $\sigma$ обозначим $\sigma \bullet e$ и определим следующим образом.

1. Если $e$ — это примитивное значение, то $\sigma \bullet e \overset{\text{def}}{=} e$.
2. $\sigma \bullet op(e_1, \dots, e_n) \overset{\text{def}}{=} op(\sigma \bullet e_1, \dots, \sigma \bullet e_n)$.
3. $\sigma \bullet union\{\langle g_1, t_1\rangle, \dots, \langle g_n, t_n\rangle\} \overset{\text{def}}{=} union\{\langle \sigma \bullet g_1, \sigma \bullet t_1\rangle, \dots, \langle \sigma \bullet g_n, \sigma \bullet t_n\rangle\}$.
4. $\sigma \bullet LI(l) \overset{\text{def}}{=} read(\sigma, \sigma \bullet l)$.

Интуитивно, $\sigma \bullet e$ — это выражение, получаемое подстановками значений из $\sigma$ в символьные ячейки $e$: первые три пункта определения сохраняют структуру $e$, а п.4 заполняет ячейку значением из $\sigma$.

**Определение 5.** *Композиция символьных куч* $\sigma$ *и* $\sigma'$ — это частичная функция $\sigma \circ \sigma' : loc \to term$, определяемая так: $(\sigma \circ \sigma')(x) \overset{\text{def}}{=}$

$$\overset{\text{def}}{=} union(\{\langle x = \sigma \bullet l, \sigma \bullet (\sigma'(l))\rangle | l \in dom(\sigma')\} \cup \langle \bigwedge_{l \in dom(\sigma')} x \neq \sigma \bullet l, \sigma(x)\rangle).$$

Композиция $\sigma \circ \sigma'$ определена на всех локациях, удовлетворяющих ограничению $x = \sigma \bullet l$, где $l \in dom(\sigma')$, и на всех локациях $dom(\sigma)$ (здесь и далее запись $\{\sigma \bullet a | a \in A\}$ сокращается как $\sigma \bullet A$). Из этого следует, что:

$$dom(\sigma \circ \sigma') = dom(\sigma) \cup \sigma \bullet dom(\sigma'). \quad (3)$$

Композиция символьных куч отражает последовательную композицию в программировании: если $\sigma_1$ — это эффект фрагмента кода A и $\sigma_2$ — эффект фрагмента кода B, тогда $\sigma_1 \circ \sigma_2$ — это эффект A;B. Интуитивно, $\sigma_1 \circ \sigma_2$ — это символьная куча, полученная заполнением символьных ячеек из $\sigma_2$ значениями из контекста $\sigma_1$ с последующей их записью в контекст $\sigma_1$.

**Пример 2.** Пусть $\sigma = \{x \mapsto 42; y \mapsto 7\}$ и $\sigma' = \{y \mapsto LI(x) - LI(y)\}$. Тогда $\sigma \circ \sigma' = \{x \mapsto 42; y \mapsto 42 - 7\}$.

**Теорема 1.** Если $\sigma$ и $\sigma'$ — символьные кучи, то $\sigma \circ \sigma'$ также символьная куча.

**Теорема 2.** Для произвольной символьной кучи $\sigma$, локации $x$ и выражения $e$ справедливо следующее:

(a) $read(\epsilon, x) = LI(x)$

(b) $\epsilon \bullet e = e$

(c) $\epsilon \circ \sigma = \sigma$

(d) $\sigma \circ \epsilon = \sigma$

Стоит отметить, что имеется некоторое сходство между чтением (опр. 3) и композицией (опр. 5): объединения осуществляют поиск $x$ среди локаций кучи. Если поиск был успешен, возвращается соответствующее (возможно изменённое) значение, в ином случае — значение по умолчанию. Воспользуемся этим сходством для определения оператора $find$, который далее используется для трансляции обобщённых куч в чистые функции.

**Определение 6.** $find(\sigma, x, \tau, d) \stackrel{\text{def}}{=}$

$$\stackrel{\text{def}}{=} union(\{\langle x = \tau \bullet l, \tau \bullet (\sigma(l))\rangle | l \in dom(\sigma)\} \cup \langle \bigwedge_{l \in dom(\sigma)} x \neq \tau \bullet l, d\rangle)$$

Теор. 2 позволяет компактно выразить $read$ и композицию куч через $find$:

$$read(\sigma, x) = find(\sigma, x, \epsilon, LI(x)) \qquad (4)$$

$$(\sigma \circ \sigma')(x) = find(\sigma', x, \sigma, \sigma(x)) \qquad (5)$$

**Теорема 3.** Для всех символьных куч $\sigma$, $\sigma'$ и символьных локаций $x$ справедливо следующее:

$$\sigma \bullet read(\sigma', x) = read(\sigma \circ \sigma', \sigma \bullet x).$$

| | |
|---|---|
| 1. **goto** {**true** -> F}<br>2. G:<br>3. x := a.Key + 5<br>4. **goto** {**true** -> Exit}<br>5. F:<br>6. a := **new** {Key = 10}<br>7. **goto** {**true** -> G}<br>8. Exit:<br>9. r := x<br>10. **halt** | $\sigma_G = \{x \mapsto (LI(LI(a).Key) + 5)\}$<br>$read(\sigma_G, x) = \sigma_G(x) = LI(LI(a).Key) + 5$<br>$\sigma_F \bullet read(\sigma_G, x) = 10 + 5$<br><br>$\sigma_F \circ \sigma_G = \{a \mapsto 0x1;\ 0x1.Key \mapsto 10;\ x \mapsto (10 + 5)\}$<br>$read(\sigma_F \circ \sigma_G, \sigma_F \bullet x) = read(\sigma_F \circ \sigma_G, x) = 10 + 5$ |

*Рис. 5. Чтение из композиции состояний*
*Fig. 5. Read from composition example*

Теорема 3 говорит о корректности чтения из композиции состояний. Например, имея состояния $\boldsymbol{\sigma_F}$ и $\boldsymbol{\sigma_G}$ исполняемых друг за другом фрагментов кода (как на рис. 5) и читая после фрагмента $\boldsymbol{G}$ переменную $\boldsymbol{x}$, можно прочитать переменную из позднего состояния $\boldsymbol{G}$, а затем заполнить результат из контекста $\boldsymbol{F}$. Однако возможно также прежде воспроизвести эффект $\boldsymbol{G}$ поверх эффекта $\boldsymbol{F}$ ($\boldsymbol{\sigma_F \circ \sigma_G}$), и прочитать $\boldsymbol{x}$ из уточнённого состояния. Теорема 3 утверждает, что результаты двух таких операций совпадут.

**Теорема 4.** Для всех символьных куч $\sigma$, $\sigma'$ и символьного выражения $e$ справедливо следующее: $(\sigma \circ \sigma') \bullet e = \sigma \bullet (\sigma' \bullet e)$.

Допустим, имеется три последовательных фрагмента кода с метками $F$, $G$ и $H$, причём каждый фрагмент заканчивается переходом на следующую метку в этом списке. Интуитивно, итоговое символьное состояние всего кода не должно зависеть от порядка применения эффектов этих фрагментов. Следующая теорема показывает, что КСП обладает указанным свойством.

**Теорема 5.** Для всех символьных куч $\sigma_1$, $\sigma_2$ и $\sigma_3$ справедливо следующее:

$$(\sigma_1 \circ \sigma_2) \circ \sigma_3 = \sigma_1 \circ (\sigma_2 \circ \sigma_3).$$

**Теорема 6.** Пусть $\Sigma$ — множество всех символьных куч. Тогда $(\Sigma, \circ)$ — моноид.

*Доказательство*. Из теоремы 1 следует замкнутость множества $\Sigma$ относительно операции $\circ$, по теореме 2 $\epsilon$ — нейтральный элемент, и по теореме 5 операция $\circ$ ассоциативна.

## 4.4 Объединение символьных куч

До сих пор мы рассматривали символьные состояния только для линейных фрагментов кода, в которых все переходы по меткам были безусловными ($\boldsymbol{goto}\,\{true \to \cdots\}$). Для более сложных состояний необходим новый оператор, позволяющий объединить несколько состояний в одно.

**Определение 7.** *Объединением* $\sigma = merge(\langle g_1, \sigma_1\rangle, \ldots, \langle g_n, \sigma_n\rangle)$ символьных куч $\sigma_1, \ldots, \sigma_n$ по непересекающимся ограничениям $g_1, \ldots, g_n$ будем называть частичную функцию с $dom(\sigma) = \bigcup_{i=1}^{n} dom(\sigma_i)$, для которой выполняется следующее: $(merge\langle g_i, \sigma_i\rangle)(x) \stackrel{\text{def}}{=} union\langle g_i, read(\sigma_i, x)\rangle$.

**Теорема 7.** Для любой символьной кучи $\sigma$ и произвольных символьных локаций $x$, $y$ справедливо следующее:

$$union(\langle x = y, read(\sigma, x)\rangle) = union(\langle x = y, read(\sigma, y)\rangle).$$

Далее покажем, что оператор $merge$ обладает интуитивными свойствами. **Теорема 8.** Для любых символьных куч $\sigma_1, \ldots, \sigma_n$ и произвольных непересекающихся ограничений $g_1, \ldots, g_n$, справедливо утверждение о том, что $merge\langle g_i, \sigma_i\rangle$ — символьная куча.

**Теорема 9.** Для любых символьных куч $\sigma_1, \ldots, \sigma_n$, произвольных непересекающихся ограничений $g_1, \ldots, g_n$ и для любых локаций $x$ справедливо следующее:

$$read(merge\langle g_i, \sigma_i\rangle, x) = union\langle g_i, read(\sigma_i, x)\rangle.$$

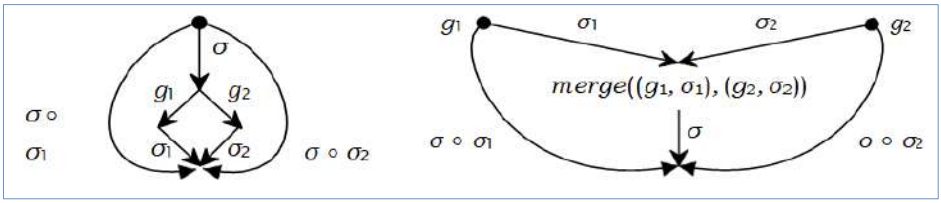


*Рис. 6. Композиция объединения куч*
*Fig. 6. Composition of heap merge*

Как уже было показано, результат вычисления символьного состояния не зависит от порядка применения эффектов. Необходимо показать, что это свойство КСП выполняется и для нового оператора объединения состояний. Для этого нужно рассмотреть два случая расстановки объединения и композиции, представленные на рис. 6.

**Теорема 10.** Для любых символьных куч $\sigma, \sigma_1, \ldots, \sigma_n$ и произвольных непересекающихся ограничений $g_1, \ldots, g_n$ выполняется следующее утверждение:

$$\sigma \circ merge(\langle g_1, \sigma_1\rangle, \ldots, \langle g_n, \sigma_n\rangle) = merge(\langle \sigma \bullet g_1, \sigma \circ \sigma_1\rangle, \ldots, \langle \sigma \bullet g_n, \sigma \circ \sigma_n\rangle).$$

Интересно, что симметричный случай гораздо сложнее. Например, рассмотрим $\sigma_1 = \{x \mapsto LI(a)\}$, $\sigma_2 = \{x \mapsto LI(b)\}$, $\sigma = \{LI(x).Key \mapsto 42\}$. Тогда:

$dom(merge(\langle g, \sigma_1\rangle, \langle \neg g, \sigma_2\rangle) \circ \sigma) =$

$= dom(merge(\langle g, \sigma_1\rangle, \langle \neg g, \sigma_2\rangle)) \cup merge(\langle g, \sigma_1\rangle, \langle \neg g, \sigma_2\rangle) \bullet dom(\sigma) =$

$= \{x, union(\langle g, LI(a).Key\rangle, \langle \neg g, LI(b).Key\rangle)\},$

что не то же самое, что

$dom(merge(\langle g, \sigma_1 \circ \sigma\rangle, \langle \neg g, \sigma_2 \circ \sigma\rangle)) = dom(\sigma_1 \circ \sigma) \cup dom(\sigma_2 \circ \sigma) =$

$= \{LI(a).Key, LI(b).Key, x\}$.

Чтобы избежать проблем такого вида, далее мы будем требовать, чтобы символьные ячейки $LI(*)$ удовлетворяли следующему дополнительному свойству: для любых непересекающихся ограничений $g_1, \dots, g_n$ и символьных локаций $x_1, \dots, x_n$ должно выполняться:

$$LI(union(\langle g_1, x_1\rangle, \dots, \langle g_n, x_n\rangle)) = union(\langle g_1, LI(x_1)\rangle, \dots, \langle g_n, LI(x_n)\rangle). \quad (6)$$

**Теорема 11.** Для любой символьной кучи $\sigma$ и произвольных непересекающихся ограничений $g_1, \dots, g_n$ , а также любых локаций $x_1, \dots, x_n$ справедливо следующее утверждение:

$$read(\sigma, union\langle g_i, x_i\rangle) = union\langle g_i, read(\sigma, x_i)\rangle.$$

Теперь необходимо сформулировать вспомогательное утверждение, которое позволит доказать симметричную теорему о композиции с объединением: $merge\langle g_i, \sigma_i\rangle \bullet e = union\langle g_i, \sigma_i \bullet e\rangle$. Однако оно не всегда верно: может случиться так, что $g_1 \vee \dots \vee g_n \neq \top$, что, в свою очередь, может привести к попытке уточнить терм в несуществующей куче. Например, можно ожидать, что уточнение терма 42 в любой куче даст 42, однако в рамках наших определений мы получим $union(\langle g_1, 42\rangle, \dots, \langle g_1, 42\rangle) = union\langle g_1 \vee \dots \vee g_n, 42\rangle \neq 42$. Чтобы указанное выше утверждение выполнялось, необходимо ограничить результат условием существования вычисления $g_1 \vee \dots \vee g_n$.

**Теорема 12.** Для любых символьных куч $\sigma_1, \dots, \sigma_n$, произвольных непересекающихся ограничений $g_1, \dots, g_n$ и некоторого выражения $e$ справедливо следующее утверждение:

$$union\langle g_1 \vee \dots \vee g_n, merge\langle g_i, \sigma_i\rangle \bullet e\rangle = union\langle g_i, \sigma_i \bullet e\rangle.$$

Таким образом, нельзя приравнять $merge(\langle g_1, \sigma_1\rangle, \dots, \langle g_n, \sigma_n\rangle) \circ \sigma$ и $merge(\langle g_1, \sigma_1 \circ \sigma\rangle, \dots, \langle g_n, \sigma_n \circ \sigma\rangle)$ как теоретико-множественные объекты. Однако следующая теорема показывает, что определение операции чтения позволяет избежать этой проблемы: в теоретико-множественном смысле кучи могут быть не равны как отображения различных множеств ключей, однако с точки зрения операции чтения они будут совпадать.

**Теорема 13.** Для любых символьных куч $\sigma, \sigma_1, \dots, \sigma_n$, произвольных непересекающихся ограничений $g_1, \dots, g_n$ и произвольной локации $x$ справедливо следующее:

$$union\langle g_1 \vee \dots \vee g_n, read(merge\langle g_i, \sigma_i\rangle \circ \sigma, x)\rangle = read(merge\langle g_i, \sigma_i \circ \sigma\rangle, x).$$

```
1. Abs:
2. goto {x >= 0 -> Exit}
3. x := -x
4. Exit: halt
```

*Листинг 2. Объединение состояний*
*Listing 2. Merge states*

При помощи операции объединения возможно описать состояние программы на лист. 2 следующим образом:

$\sigma_- = \{x \mapsto -LI(x)\}$

$\sigma = merge(\langle LI(x) \geq 0, \epsilon\rangle, \langle \neg(LI(x) \geq 0), \sigma_-\rangle)$

$$read(\sigma, x) \overset{\text{Teop.11}}{=} union(\langle LI(x) \geq 0, read(\epsilon, x)\rangle, \langle \neg(LI(x) \geq 0), read(\sigma_-, x)\rangle) = \\ = union(\langle LI(x) \geq 0, LI(x)\rangle, \langle \neg(LI(x) \geq 0), -LI(x)\rangle).$$

## 4.5 Запись в символьную кучу

До сих пор были рассмотрены операции с кучами как с готовыми объектами. Для того, чтобы строить кучу из пустого состояния $\epsilon$, необходима операция *записи* в символьную память. Для её определения воспользуемся следующим сокращением: $ite(c, a, b) \overset{\text{def}}{=} union(\langle c, a\rangle, \langle \neg c, b\rangle)$.

**Определение 8.** *Запись символьного значения $v$ в символьную локацию $y$ символьной кучи $\sigma$ — это символьная куча $write(\sigma, y, v)$, такая что для всех $x \in dom(write(\sigma, y, \cdot)) = dom(\sigma) \cup \{y\}$, $(write(\sigma, y, v))(x) \stackrel{\text{def}}{=} ite(x = y, v, \sigma(x))$.*

Заметим, что инвариант кучи (1) для записей выполняется тривиально. Следующие теоремы показывают, что операция записи сохраняет свойство композициональности относительно других операций.

**Теорема 14.** Для любой символьной кучи $\sigma$, произвольных символьных локаций $x$, $y$ и любого символьного выражения $v$ справедливо следующее:

$$read(write(\sigma, y, v), x) = ite(x = y, v, read(\sigma, x)).$$

**Теорема 15.** Для любых символьных куч $\sigma$, $\sigma'$, произвольной символьной локации $y$ и произвольного символьного выражения $v$ справедливо следующее:

$$\sigma \circ write(\sigma', y, v) = write(\sigma \circ \sigma', \sigma \bullet y, \sigma \bullet v).$$

**Теорема 16.** Для любых символьных куч $\sigma_1, \ldots, \sigma_n$, любых непересекающихся ограничений $g_1, \ldots, g_n$, и произвольной символьной локации $y$ и символьного выражения $v$ справедливо следующее:

$$write(merge\langle g_i, \sigma_i \rangle, y, v) = merge\langle g_i, write(\sigma_i, y, v) \rangle.$$

## *5. Исчисление символьных куч*

Представленные операторы КСП уже позволяют описывать символьные состояния произвольных фрагментов кода без циклов в графе потока управления. Однако идея подстановки из контекстной кучи позволяет нам пойти дальше и определить исчисление, описывающее исполнение произвольных императивных программ с динамической памятью.

### 5.1 Обобщённые символьные кучи

Для начала определим формальный язык $Heap$ нашего исчисления. Термы языка $Heap$ будем называть *обобщёнными кучами*. Напомним, что $\Sigma$ — это множество всех символьных куч, которые были определены в предыдущем разделе.

Обобщённая куча может быть либо обычной символьной кучей (из $\Sigma$), которую мы будем называть *определённой*, либо объединением обобщённых куч по непересекающимся ограничениям, либо композицией обобщённых куч, либо записью в обобщённую кучу, либо неподвижной точкой цикла в графе потока управления, которую мы будем называть *рекурсивным состоянием* (рис. 7).

$$\begin{aligned} Heap ::= \ & \Sigma \\ & \mid Heap \circ Heap \\ & \mid merge(\langle guard, Heap \rangle^*) \\ & \mid write(Heap, loc, term) \\ & \mid Rec(id) \end{aligned}$$

*Рис. 7. Обобщённые кучи*
*Fig. 7. Generalized heaps*

Интуитивно, рекурсивные состояния программы — это состояния, зависящие от самих себя, т.е. чтение из которых требует предыдущую версию того же состояния. В общем случае такое состояние нельзя выразить в виде конечной композиции других состояний, поэтому для них вводится новый символ. Идентификатор $id$ в $Rec(id)$ уникальным

образом описывает цикл в графе потока управления: он состоит из метки из исходного кода и некоторого.

Чтобы адаптировать операции КСП, необходимо расширить синтаксис символьных выражений, как показано на рис. 8.

$$
\begin{aligned}
term &::= \ arith \mid loc \\
arith &::= \ \mathbb{N} \mid arith \pm arith \mid -arith \mid LI^{arith}(\boldsymbol{Heap}, loc) \\
&\quad \mid union^{arith}(\langle guard, arith\rangle^*) \\
loc &::= \ null \mid 0x[0-9]^+ \mid ident \\
&\quad \mid loc.FieldName \mid LI^{loc}(\boldsymbol{Heap}, loc) \\
&\quad \mid union^{loc}(\langle guard, loc\rangle^*) \\
guard &::= \ \top \mid \bot \mid \neg guard \mid guard \wedge guard \mid guard \vee guard \\
&\quad \mid arith = arith \mid arith < arith \mid loc = loc
\end{aligned}
$$

*Рис. 8. Грамматика обобщённых символьных выражений*
*Fig. 8. Symbolic generalized terms*

На чтение из *определённой* кучи $read(\sigma, x) = LI(x)$ можно смотреть как на следующее сообщение: «в $\sigma$ недостаточно информации, чтобы узнать $x$, — требуется дополнительный контекст». Сама операция чтения построена таким образом, что мы можем однозначно сказать, было ли успешно чтение $x$ из $\sigma$, — и если было, то предъявить результат. При чтении из *обобщённой* кучи, например, $read(Rec(F), x)$, могут возникнуть сложности. Если циклический фрагмент кода по метке $F$ меняет содержимое $x$, то мы не можем его корректно прочитать — для этого необходимо заранее знать, сколько раз исполнится $F$, что в общем случае невозможно. Очевидно также, что мы не можем отбросить $Rec(F)$ и вернуть просто $LI(x)$.

Таким образом, основная идея расширения термов состоит в «запоминании» источника каждого символьного значения (выделено жирным на рис. 8). Такое расширение является корректным, поскольку старые символьные значения $LI(x)$ теперь станут $LI(\epsilon, x)$. Уточнение также может быть расширено: $\tau \bullet LI(\sigma, x) = read(\tau \circ \sigma, \tau \bullet x)$. Это не нарушит свойства КСП, так как по теор. 3, мы имеем $\tau \bullet read(\sigma, x) = read(\tau \circ \sigma, \tau \bullet x)$.

## 5.2 Правила редукции

Правила редукции символьных куч представлены на рис. 9. Буквами $H$ обозначены элементы языка $Heap$, буквами $t$ — символьные термы. $H[A/X]$ означает одновременную подстановку $A$ вместо $X$ в обобщённую кучу $H$; здесь $A$ и $X$ могут быть как обобщёнными кучами, так и символьными термами.

Корректность всех правил, кроме (8), обоснована в разд. 4. Правила (7) и (8) представляют собой аналог $\alpha$-конверсии и $\beta$-редукции в $\lambda$-исчислении. $Body(x)$ представляет собой описание поведения участка кода, которому соответствует обобщённая куча $Rec(x)$. Если посмотреть на исчисление символьных куч как на некоторый язык программирования, то $Rec(x)$ соответствовало бы имени функции, а $Body(x)$ — её телу. Обобщённую кучу назовём *нередуцируемой*, если к ней нельзя применить ни одного правила за исключением (7). $H \to^n H'$ означает, что куча $H$ редуцируется в $H'$ за $n$ шагов. $H \to^* H'$ означает, что существует $n \geq 0$, что $H \to^n H'$.

Теперь продемонстрируем, как можно представлять символьные состояния циклических фрагментов кода на примере с лист. 3. Так как выход из цикла зависит от получаемого списка $p$, и его длина заранее неизвестна, то состояние этого фрагмента нельзя представить в виде какой-либо конечной композиции определённых символьных

состояний. Однако можно построить обобщённую кучу $Body(Inc)$, описывающую поведение метки $Inc$.

$$\frac{t_1 = t_2}{H \to H[t_2/t_1]}\ (7)$$

$$H \to H[Body(id)/Rec(id)]\ (8)$$

$$\epsilon \circ H \to H$$

$$H \circ \epsilon \to H$$

$$H_1 \circ (H_2 \circ H_3) \to (H_1 \circ H_2) \circ H_3$$

$$H \circ merge\langle g_i, H_i\rangle \to merge\langle H \bullet g_i, H \circ H_i\rangle$$

$$merge\langle g_i, H_i\rangle \circ H \to merge\langle g_i, H_i \circ H\rangle$$

$$H \circ write(H', x, v) \to write(H \circ H', H \bullet x, H \bullet v)$$

$$write(merge\langle g_i, H_i\rangle, x, v) \to merge\langle g_i, write(H_i, x, v)\rangle$$

$$\frac{g \text{ невыполнимо}}{merge(\langle g, H\rangle \cup X) \to merge(X)}$$

$$\frac{g \text{ общезначимо}}{merge\langle g, H\rangle \to H}$$

*Рис. 9. Правила редукции*
*Fig. 9. Reduction rules*

```
1. Inc:
2. goto {p = null -> Exit}
3. p.Key := p.Key + 1
4. p := p.Next
5. goto {true -> Inc}
6. Exit: halt
```

*Листинг 3. Фрагмент кода с циклом в графе потока управления*
*Listing 3. Code snippet with a cycle in a control flow graph*

Пусть $\sigma_0$ — некоторая символьная куча, представляющая начальное состояние исполнения, а обобщённая куча $Rec(Inc)$ соответствует метке $Inc$. Тогда поведение всего кода в лист. 3 на состоянии $\sigma_0$ будет описываться обобщённой кучей $\sigma_0 \circ Rec(Inc)$. Применение правил редукции к $\sigma_0 \circ Rec(Inc)$ будет соответствовать вычислению кода с метки $Inc$ на состоянии $\sigma_0$. Покажем это на примере.

Пусть $\sigma_0 = \{p \mapsto 0x1, 0x1.Key \mapsto 10, 0x1.Next \mapsto 0x2, 0x2.Key \mapsto 20,\ 0x2.Next \mapsto null\}$.

Опишем теперь поведение циклического региона, помеченного $Inc$. В начале исполнения происходит ветвление по условию $p = null$. Если $p \neq null$, то стр. 3-5 увеличивают значение ключа в узле связного списка и переходят к следующему элементу. Поведение этого участка кода можно описать символьным объединением двух эффектов: пустого эффекта $\epsilon$ (т.к. переход на стр. 2 не меняет состояния) и эффекта $\sigma$ нерекурсивного кода на стр. 3-4, где $\sigma = \{LI(p).Key \mapsto LI(LI(p).Key) + 1, p \mapsto LI(LI(p).Next)\}$.

Таким образом, поведение региона $Inc$ описывается обобщённой кучей

$$Body(Inc) = merge(\langle LI(p) = null, \epsilon\rangle, \langle LI(p) \neq null, \sigma \circ Rec(Inc)\rangle).$$

Теперь опишем процесс редукции кучи $\sigma_0 \circ Rec(Inc)$.

$\sigma_0 \circ Rec(Inc) \to \sigma_0 \circ Body(Inc) \to$

$merge(\langle \sigma_0 \bullet (LI(p) = null), \sigma_0 \circ \epsilon\rangle, \langle \sigma_0 \bullet (LI(p) \neq null), \sigma_0 \circ (\sigma \circ Rec(Inc))\rangle) \to^4$

$merge(\langle 0x1 = null, \sigma_0\rangle, \langle 0x1 \neq null, (\sigma_0 \circ \sigma) \circ Rec(Inc)\rangle) \to^2 \sigma_1 \circ Rec(Inc) \to^2$

$merge(\langle\sigma_1 \bullet (LI(p) = null), \sigma_1 \circ \epsilon\rangle, \langle\sigma_1 \bullet (LI(p) \neq null), \sigma_1 \circ (\sigma \circ Rec(Inc))\rangle) \rightarrow^4$
$merge(\langle 0x2 = null, \sigma_1\rangle, \langle 0x2 \neq null, (\sigma_1 \circ \sigma) \circ Rec(Inc)\rangle) \rightarrow^2 \sigma_2 \circ Rec(Inc) \rightarrow^2$
$merge(\langle\sigma_2 \bullet (LI(p) = null), \sigma_2 \circ \epsilon\rangle, \langle\sigma_2 \bullet (LI(p) \neq null), \sigma_2 \circ (\sigma \circ Rec(Inc))\rangle) \rightarrow^4$
$merge(\langle null = null, \sigma_2\rangle, \langle null \neq null, (\sigma_2 \circ \sigma) \circ Rec(Inc)\rangle) \rightarrow^2 \sigma_2$

Здесь

$\sigma_1 \stackrel{\text{def}}{=} \sigma_0 \circ \sigma = \{p \mapsto 0x2, 0x1.Key \mapsto 11, 0x1.Next \mapsto 0x2,$
$0x2.Key \mapsto 20, 0x2.Next \mapsto null\},$

$\sigma_2 \stackrel{\text{def}}{=} \sigma_1 \circ \sigma = \{p \mapsto null, 0x1.Key \mapsto 11, 0x1.Next \mapsto 0x2,$
$0x2.Key \mapsto 21, 0x2.Next \mapsto null\}.$

Обобщённая куча $\sigma_2$ не редуцируема (заметим, что нередуцируемым будет любой элемент Σ). $\sigma_0$ и $\sigma_1$ представляют собой состояния изначальной императивной программы в процессе её исполнения, а $\sigma_2$ — её конечное состояние (при запуске на $\sigma_0$).

Исчисление символьных куч позволяет описывать произвольные поведения программ с динамической памятью без потери информации.

## *6. Композициональное символьное исполнение*

В данном разделе предложен алгоритм композиционального символьного исполнения, который позволяет автоматически проверять достижимость ошибок при произвольном графе потока управления. Он основан на подходе символьного исполнения программ [22].

### 6.1 Метод описания путей в графе потока управления

Этот метод является основным в предлагаемом алгоритме, т.к. выполняет описание *всех* путей в графе потока управления. Данный метод, в частности, позволяет *автоматически* строить обобщённые символьные кучи $Body(id)$, описывающие поведения циклических фрагментов программы.

Прямолинейным способом построения обобщённых куч по императивной программе является введение куч $Rec(l)$ для каждой инструкции $l$ и взятие композиции со всеми $Rec(l')$, в которые есть переход из инструкции $l$. Однако такой подход порождает слишком большую систему взаимно-рекурсивных определений: фактически, количество символов-абстракций $Rec(l)$ было бы равно количеству инструкций в программе. В данном разделе описывается метод описания всех возможных путей исполнения программы через введение меньшего числа символов-абстракций.

**Определение 9.** *Вершинами* $V_G$ графа потока управления $G$ будут номера $l$ инструкций, а *рёбрами* $E_G$ — пары номеров $(l_x, l_y)$, указывающие на возможность передачи управления от инструкции $l_x$ к инструкции $l_y$. В графе потока управления существует *начальная* вершина, которая соответствует первой инструкции программы и в которую не ведёт ни одно ребро. Согласно грамматике демо-языка (см. рис. 1), большинство рёбер будут иметь вид $(l, succ(l))$. Однако благодаря оператору **goto** возможны переходы к произвольным инструкциям, кроме начальной.

**Определение 10.** Проведём обход графа $G$ в глубину и для каждой вершины $v$ вычислим время «выхода» $time(v)$ из обхода. Вершина $l$ называется *рекурсивной*, если существует ребро $(l', l)$ такое, что $time(l') \geq time(l)$. Множество рекурсивных вершин будем обозначать $RV$.

Для описания путей необходимо ввести операцию *конкатенации* двух путей в графе. Неформально, конкатенация путей $p_1$ и $p_2$ — это путь $p_1 \circ p_2$, который содержит рёбра пути $p_1$, за которыми следуют ребра пути $p_2$. Конкатенация двух множеств путей $P_1$ и $P_2$

определяется через конкатенацию двух путей: $P_1 \circ P_2 = \{p_1 \circ p_2 | p_1 \in P_1, p_2 \in P_2\}$. Символ $\varepsilon$ означает *пустой путь*, а символ $\cup$ — объединение множеств путей.

Предлагаемый метод позволяет описывать в точности все пути в произвольном графе потоке управления при помощи рекурсивных символов и их рекурсивных описаний, на базе которых далее будут строиться обобщённые символьные кучи.

$$\Pi(u,v,D) = \bigcup_{(u,v)\in E_G} \{(u,v)\} \cup \bigcup_{\substack{t\notin RV\cup\{v\} \\ (u,t)\in E_G}} (u,t) \circ \Pi(t,v,D) \cup \bigcup_{\substack{t\in RV\setminus(D\cup\{v\}) \\ (u,t)\in E_G}} (u,t) \circ Rec(t, D\cup\{t\}) \circ \Pi(t,v,D\cup\{t\})$$

$$Rec(u,D) = \{\varepsilon\} \cup \Pi(u,u,D) \circ Rec(u,D)$$

Символом $\Pi(u,v,D)$ обозначим множество путей из вершины $u$ в вершину $v$, параметризованное множеством *пройденных* рекурсивных вершин $D$. Множество $D$ ограничивает переходы по рёбрам: ребро $(l_x, l_y)$ является «допустимым», если оно ведёт в конечную вершину (т.е. $l_y = v$) или $l_y \neq v$ и вершина $l_y$ не была ещё посещена (т.е. $l_y \notin D$). *Рекурсивным символом* $Rec(u,D)$ обозначим множество путей-циклов из вершины $u$ в вершину $u$ с множеством $D$, имеющим тот же смысл, как и для $\Pi(u,v,D)$.

Интуитивно, $\Pi(u,v,D)$ соответствует символьным кучам, полученным в результате символьного исполнения программы от инструкции с номером $u$ до инструкции с номером $v$, когда исполнение не посещало инструкции с номерами из множества $D\backslash\{v\}$. В свою очередь, *рекурсивный символ* $Rec(u,D)$ соответствует *обобщённой символьной куче* $Rec(id)$, у которой уникальным идентификатором $id$ является пара $(u,D)$, а его описание — это обобщённая символьная куча $Body(u,D)$. Кроме того, оператор $\circ$ соответствует операции *композиции* состояний, символ $\cup$ — операции *объединения* состояний ($merge$), а $\varepsilon$ — пустой куче $\epsilon$.

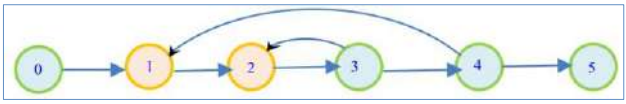


*Рис. 10. Пример графа потока управления с вложенными циклами*
*Fig. 10. An example of a control flow graph with nested loops*

Покажем на примере, как можно описать все пути в графе потока управления при помощи итеративного построения $\Pi$ и $Rec$. На рис. 10 представлен пример графа потока управления программы с вложенными циклами. Для простоты изложения номер вершины $v$ равен времени $time(v)$. На рис. 10 вершины 1 и 2 являются рекурсивными. Ниже представлено описание всех путей в графе из 0 в 5, использующее рекурсивные символы.

$\Pi(0,5,\varnothing) = (0,1) \circ Rec(1,\{1\}) \circ (1,2) \circ Rec(2,\{1,2\}) \circ (2,3) \circ (3,4) \circ (4,5)$

$Rec(1,\{1\}) = (1,2) \circ Rec(2,\{1,2\}) \circ (2,3) \circ (3,4) \circ (4,1) \circ Rec(1,\{1\}) \cup \{\varepsilon\}$

$Rec(2,\{1,2\}) = (2,3) \circ (3,2) \circ Rec(2,\{1,2\}) \cup \{\varepsilon\}$

$\Pi(0,5,\varnothing)$ обозначает все пути из вершины 0 в вершину 5. При переходе по ребру $(0,1)$ метод попадает в вершину 1. Так как она *рекурсивная* (поскольку лежит в $RV$), метод вводит для неё символ $Rec(1,\{1\})$ и начинает создавать *рекурсивное* описание этого символа.

Это описание начинается с перехода в вершину 2 и введения рекурсивного символа для неё. Поскольку были пройдены обе *рекурсивные* вершины, то при создании символа $Rec(2,\{1,2\})$ множество $D = \{1,2\}$. Рекурсивное определение для $Rec(2,\{1,2\})$ выглядит следующим образом: все пути из 2 в 2, не проходящие через $D = \{1,2\}$ в середине пути,

— это повторения пути $(2,3) \circ (3,2)$. Кроме того, любое множество $Rec(\cdot)$ путей из себя в себя содержит пустой путь $\varepsilon$.

Далее продолжается описание $Rec(1,\{1\})$: после перехода из 1 в 2 происходит конкатенация пути $(1,2)$ и путей $Rec(2,\{1,2\})$ из 2 в себя, а к множеству $D$ добавляется вершина 2. После этого путь, возвращающийся в вершину 1, очевиден: $(2,3) \circ (3,4) \circ (4,1)$.

Затем процесс возвращается к построению $\Pi(0,5,\varnothing)$. После перехода по ребру $(0,1)$ и создания символа $Rec(1,\{1\})$ к множеству D добавляется вершина 1. Затем происходит переход по ребру $(1,2)$ и добавление соответствующего символа $Rec(2,\{1,2\})$, а также к множеству $D$ добавляется 2. Поскольку описание для символа $Rec(2,\{1,2\})$ уже было построено при построении описания $Rec(1,\{1\})$, то метод не будет строить его заново. Далее следует тривиальный путь до вершины 5: $(2,3) \circ (3,4) \circ (4,5)$.

Таким образом, метод позволяет описывать все пути в графе потока управления и только их. Формальное доказательство этого факта приведено в [27].

## 6.2 Алгоритм композиционального символьного исполнения

Благодаря соответствию между рекурсивными символами и их описаниями, с одной стороны, и обобщёнными кучами $Rec(id)$ и $Body(id)$, с другой стороны, можно получить *алгоритм автоматической проверки достижимости ошибок в программах с динамической памятью и произвольными графами потока управления*, не раскручивающий отношение перехода.

Предлагаемый алгоритм использует символьное исполнение и операции над символьными кучами (см. лист. 4). Также он использует оракул $SAT$, проверяющий достижимость веток исполнения. Для данного алгоритма важно понятие *состояния исполнения*.

**Определение 10.** *Состояние исполнения* — это кортеж $(l, pc, \sigma, D)$, где $l$ — номер инструкции, $pc$ – *условие пути* (path condition — символьная формула, описывающая ограничения на достижимость инструкции), $\sigma$ — символьная куча, $D$ — множество посещённых рекурсивных вершин.

Важнейшей функцией алгоритма является $Exec$, которая может быть вызвана либо из *начальной* (стр. 2), либо из *рекурсивной* вершины (стр. 51). В первом случае её результатом является символьная куча, соответствующая путям исполнения, которые привели к инструкциям $\boldsymbol{halt}$ (стр. 10), а также выводится множество путей, приводящих к ошибкам (стр. 53). Во втором случае результатом является символьная куча $Body(l_0, D_0)$, описывающая поведения циклического участка графа потока управления. В стр. 37-40 происходит добавление *обобщённого состояния*, представляющего собой композицию *построенного состояния* $\sigma'$ *с рекурсивным состоянием* $Rec(l_0, D_0)$. В конце функции (стр. 52) добавляется завершающее состояние для случая, когда исполнение не вернулось в *рекурсивную* вершину $l_0$, соответствующее *пустому пути* $\varepsilon$ из определения рекурсивных символов в методе описания путей в графе.

Алгоритм выбирает следующее состояние исполнения из рабочего множества ($pickNext$), затем исполняет соответствующую инструкцию, порождая новые состояния исполнения (стр. 9-35), и добавляет их в рабочее множество, объединяя те состояния исполнения, у которых равны номера инструкций $l$ и совпадают множества *посещённых* рекурсивных вершин $D$ (стр. 47-50). Стоит отметить, что предлагаемый алгоритм не раскручивает циклы, а вводит рекурсивные состояния $Rec(l_0, D_0)$ (стр. 12) и обобщённые кучи $Body(l_0, D_0)$ (стр. 46) для их описания. Все обобщённые кучи $Body(\cdot,\cdot)$ используются для определения выполнимости ограничений пути (стр. 21, 23, 31, 34).

```
1  ∀ l ∈ RV, ∀ D  Body(l, D) ← ϵ;
2  return Exec(start, ∅);
3  Function Exec(l_O : Vertex, D_O : Vertex set)
4      pc_r, σ_r ← (⊥, ϵ);
5      W ← {(l_O, ⊤, ϵ, D_O)}; Errors ← ∅;
6      while W ≠ ∅ do
7          (l, pc, σ, D), W ← pickNext(W);
8          S ← ∅;
9          switch instr(l)
10             case halt:
11                 if l_O ∉ RV then
12                     pc_r ← pc_r ∨ pc;
13                     σ_r ← merge((pc_r , σ_r ), (pc,σ));
14             case fail:  Errors ← Errors ∪ {pc};
15             case ident := expression
16                 σ, value ← Eval(σ,expression);
17                 S ← {(succ(l), pc, write(σ, ident, value))};
18             case Location.field := expression
19                 σ, value ← Eval(σ,expression);
20                 σ, loc ← Eval(σ,Location);
21                 if SAT(Body, σ, pc ∧ loc ≠ null) then
22                     S ← {(succ(l), pc ∧ loc ≠ null, write(σ, loc.field, value))};
23                 if SAT(Body, σ, pc ∧ loc = null) then
24                     Errors ← Errors ∪ {pc ∧ loc = null};
25             case label : statement
26                 S ← {(succ(l), pc, σ)};
27             case goto labels
28                 guard_succ ← ⊤;
29                 forall (expression → l') ∈ labels do
30                     σ, guard ← Eval(σ, expression);
31                     if SAT(Body, σ, pc ∧ guard ∧ guard_succ) then
32                         S ← S ∪ {(l', pc ∧ guard ∧ guard_succ, σ)};
33                     guard_succ ← guard_succ ∧ ¬guard;
34                 if SAT(Body, σ, pc ∧ guard_succ) then
35                     S ← S ∪ {(succ(l), pc ∧ guard_succ,σ)};
36         forall (l', pc', σ') ∈ S do
37             if l' = l_O then
38                 σ' ← σ' ∘ Rec(l_O, D_O);
39                 pc_r, σ_r ← (pc_r ∨ pc', merge ((pc_r, σ_r), (pc' , σ')));
40                 continue;
41             elif l' ∈ D then continue;
42             elif l' ∈ RV then
43                 D ← D ∪ {l'};
44                 σ' ← σ' ∘ Rec(l', D);
45                 if Body(l', D) = (⊥, ϵ) then
46                     Body(l', D) ← Exec(l', D);
47             if ∃(l'', pc'', σ'', D'') ∈ W : l' = l'' ∧ D =D'' then
48                 W ← W \ {(l'', pc'', σ'', D'')};
49                 W ← W ∪ {(l', pc' ∨ pc'', merge( (pc', σ'), (pc'', σ'')) , D)};
50             else W ← W ∪ {(l', pc', σ', D)};
51     if l_O ∈ RV then
52         return merge (pc_r, σ_r), (¬pc_r, ϵ) ;
53     print Errors;
54     return σ_r
```

*Листинг 4. Алгоритм композиционального символьного исполнения*
*Listing 4. Compositional symbolic execution algorithm*

Функция $Eval(\sigma, expression)$ вычисляет выражения, трактуя арифметические и булевы операции стандартным образом и читая переменные из состояния $\sigma$.

Стоит заметить, что в качестве побочного эффекта $Eval(\sigma, expression)$ может добавить к множеству $Errors$ новое условие пути, защищающее обращение к нулевому адресу. Для вычисления $Eval(\sigma, \textbf{\textit{new}}\{Field_i \rightarrow Expr_i\})$ будет создан новый уникальный адрес

$0xNNN$[3], все $Expr_i$ вычислятся в $v_i$ и состояние будет итеративно обновлено: $\sigma' = write(\sigma, 0xNNN.Field_i, v_i)$. Функция $Eval$ вернёт новое состояние $\sigma'$ и адрес созданного объекта $0xNNN$. Например, для фрагмента на лист. 5 может быть получено следующее новое состояние:

$$0x40.K \mapsto 30 \quad 0x41.K \mapsto 10 \quad 0x42.K \mapsto 50$$
$$0x40.L \mapsto 0x41 \quad 0x41.L \mapsto null \quad 0x42.L \mapsto null$$
$$0x40.R \mapsto 0x42 \quad 0x41.R \mapsto null \quad 0x42.R \mapsto null$$

```
1.  x = new {K = 30;
2.           L = new {K = 10; L = null; R = null};
3.           R = new {K = 50; L = null; R = null}}
```

*Листинг 5. Программа, выделяющая память*
*Listing 5. Heap-allocating program*

## 6.3 Корректность алгоритма композиционального символьного исполнения

**Определение 12.** *Замкнутым* назовём терм, не содержащий $LI(\cdot)$.

*Конкретная куча* — это (тотальное) отображение из замкнутых локаций в замкнутые термы. Множество конкретных куч обозначим за $\Sigma_G$.

Конкретная куча представляет собой состояние динамической памяти при конкретном исполнении программы. Заметим, что $(\Sigma_G, \circ)$ — правый идеал в моноиде $(\Sigma, \circ)$: если $\sigma \in \Sigma_G, \tau \in \Sigma$, то $\sigma \circ \tau \in \Sigma_G$. Этот факт позволяет легко доказать следующее утверждение.

**Утверждение 2.** Если для $\sigma \in \Sigma_G$ и обобщённой кучи $H$, $\sigma \circ H \to^* H'$ для некоторой нередуцируемой $H' \in Heap$, то $H' \in \Sigma_G$.

Пусть $T: \mathbb{N} \times \Sigma_G \to \mathbb{N} \times \Sigma_G$ — отношение перехода некоторой программы на демо-языке (т.е. отображение, которое номеру инструкции и состоянию программы сопоставляет следующую инструкцию и состояние, полученное исполнением входной инструкции на входном состоянии). Обозначим $T^n(l, \sigma) \stackrel{\text{def}}{=} \underbrace{T(\ldots T}_{n \text{ раз}}(l, \sigma))$.

Следующая теорема (которую мы оставляем без доказательства) говорит о корректности алгоритма на лист. 4.

**Теорема 17.** Пусть $T$ — отношение перехода программы $P$ на демо-языке, $l_0$ — номер начальной инструкции, $F$ — множество номеров инструкций ***halt*** и ***fail*** в программе, $\sigma_0 \in \Sigma_G$, $H \stackrel{\text{def}}{=} Exec(l_0, \varnothing)$. Также допустим, что оракул $SAT$ всегда отвечает правильно. Тогда $T^n(l_0, \sigma_0) = (f, \tau)$ для некоторых $n \in \mathbb{N}, f \in F, \tau \in \Sigma_G$ т. и т. т., к. $\sigma_0 \circ H \to^* \tau$.

## *7. Трансляция символьных куч в чистые функции*

Для проверки достижимости некоторого пути исполнения программы, алгоритм из разд. 6 обращается к функции-оракулу $SAT$. В данном разделе мы определяем $SAT(Body, \sigma, g)$ и тем самым завершаем построение композициональной процедуры верификации. Мы сведём задачу выполнимости ограничения $g$ к задаче доказательства безопасности функциональной программы без эффектов, состоящей из чистых функций второго порядка[4].

[3] В текущем изложении некорректно обрабатывается случай с выделением объекта в циклическом регионе; для корректной работы определение уточнения должно быть изменено: $\sigma \bullet 0xNNN$ должно порождать *новый* адрес $0xMMM$

[4] Функцией *первого порядка* называется функция, которая не принимает в аргументы другие функции. Функцией *второго порядка* называется функция, которая принимает в аргументы функции только первого порядка — и не выше.

## 7.1 Оператор Find

Этот оператор, определённый в разд. 4.3, играет важную роль в трансляции обобщённых куч в чистые функции, обобщая чтение и композицию символьных куч. Сформулируем его основное свойство.

**Утверждение 3.** Для всех определённых символьных куч $\sigma$, $\sigma'$, $\tau$ таких, что для каждого символьного выражения $e$, выполняется $(\tau \circ \sigma) \bullet e = \tau \bullet (\sigma \bullet e)$, и всех символьных выражений $x \in loc$, $d \in term$, верно следующее:

$$\tau \bullet find(\sigma', x, \sigma, d) = find(\sigma', \tau \bullet x, \tau \circ \sigma, \tau \bullet d).$$

Далее, можно заметить, что:

$$read(\sigma, x) = find(\sigma, x, \epsilon, LI(x)) = find(\sigma, x, \epsilon, read(\epsilon, x)),$$
$$(\sigma \circ \sigma')(x) = find(\sigma', x, \sigma, \sigma(x)) = find(\sigma', x, \sigma, read(\sigma, x)).$$

Это даёт возможность определить оператор $find$ для обобщённых куч следующим образом (обозначив прежний $find$ как $find^{\Sigma}$):

$$find(\sigma, x, \tau) \stackrel{\text{def}}{=} \begin{cases} find^{\Sigma}(\sigma, x, \tau, find(\tau, x, \epsilon)), & \text{если } \sigma \in \Sigma \\ LI(\tau \circ \sigma, x), & \text{иначе} \end{cases} \quad (10)$$

## 7.2 Трансляция обобщённых куч в функции второго порядка

Будем говорить, что символьный терм $t$ находится в *нормальной форме*, если он содержит объединения только на верхнем уровне, т.е. $t = union(\langle g_1, t_1 \rangle, \ldots, \langle g_n, t_n \rangle)$ и ни одно из ограничений $g_i$ и ни один из термов $t_i$ не содержат внутри $union$. Будем также говорить, что *ограничение* находится в нормальной форме, если оно не содержит объединений. Каждое символьное выражение может быть нормализовано: по утв. 1 и (6), вложенные объединения могут быть линеаризованы. Если $t$ не содержит объединений, тогда его нормальной формой будем называть $union\langle \top, t \rangle$. По определению, ограничение $g \equiv union(\langle g_1, c_1 \rangle, \ldots, \langle g_n, c_n \rangle)$ может быть переписано в $(g_1 \wedge c_1) \vee \ldots \vee (g_n \wedge c_n)$.

Рассмотрим символьную ячейку $LI(\sigma, x)$. Заметим, что такие ячейки с $\sigma \neq \epsilon$ появляются только в последней ветке определения (10), т.е. можно рассматривать $LI(\sigma, x)$ как $find(\sigma, x, \epsilon)$. Уточнение такого выражения в контексте $\tau$ даст $\tau \bullet find(\sigma, x, \epsilon) \stackrel{\text{утв.3}}{=} find(\sigma, \tau \bullet x, \tau)$. Это даёт возможность транслировать символьные выражения в функции второго порядка.

Далее, с помощью $\tau$ обозначим функцию первого порядка «чтение из контекстной кучи». Преобразование символьного выражения $e$ в выражение функционального языка при контекстной куче $\tau$ обозначим как $[\![e]\!]_{\tau}$. Это преобразование состоит из трёх следующих шагов.

1. Нормализация $e$ и преобразование верхнеуровневого объединения в конструкцию ветвления.
2. Замена всех ячеек $LI(\sigma, x)$ на $find(\sigma, [\![x]\!]_{\tau}, \tau)$.
3. Специализация оператора $find$ согласно правилам (10). На этом шаге все термы вида $find(\sigma, x, \tau)$ транслируются в применения функций второго порядка $\text{find}_{\sigma}$. Телом функции $\text{find}_{\sigma}$ будет результат применения этих трёх шагов к соответствующему правилу (10). При появлении композиции $\sigma \circ \sigma'$ контекстное состояние становится частичным применением $\text{find}_{\sigma}$ к текущему контекстному состоянию $\tau$.

Вместо формального описания целевого функционального языка программирования и алгоритма трансляции мы продемонстрируем процесс трансляции на примере. Допустим, необходимо ответить на запрос $SAT(Body, Rec(f), LI(\epsilon, a) * 3 < 17)$. Пусть $Body(f) = merge(\langle c, \epsilon \rangle, \langle \neg c, \sigma \circ Rec(f) \rangle)$, где $\sigma$ — это некоторая обобщённая куча. Тогда

необходимо проверить выполнимость ограничения $g = (Rec(f) \bullet (LI(\epsilon, a) * 3) < 17) = (LI(Rec(f), a) * 3 < 17)$.

Сначала определим контекстную функцию первого порядка $\tau_0$, которая будет принимать адрес и выполнять ленивое инстанцирование символьных локаций, т.е. возвращать недетерминированные значения. Далее, вычислим $[\![g]\!]_{\tau_0}$.

Первый шаг не порождает условных конструкций. После второго шага выражение $g$ становится следующим: $find(Rec(f), a, \tau_0) * 3 < 17$. Третий шаг порождает новую функцию второго порядка $\text{find}_{Rec(f)}$. Таким образом, закодированное значение $g$ будет: $[\![g]\!]_{\tau_0} = (find_{Rec(f)}\ \tau_0\ a) * 3 < 17$.

Проверить выполнимость $g$ — это то же самое, что проверить безопасность программы “assert(¬ g)”.

Теперь мы должны задать тела полученных функций $find$. Пусть $\text{find}_f$ принимает контекстную функцию первого порядка $\tau$ и локацию $x$. Тело функции $\text{find}_{Rec(f)}$ мы получим, применяя шаги 1-3 к $Body(f)$:

$$find(merge(\langle c, \epsilon\rangle, \langle \neg c, \sigma \circ Rec(f)\rangle), x, \tau) \overset{(10)}{=} \\ = ite(\tau \bullet c, find(\epsilon, x, \tau), find(\sigma \circ Rec(f), x, \tau))$$

Это объединение будет нормализовано и транслируется в ветвление в теле $\text{find}_{Rec(f)}$; ленивые ячейки в $c$ заменятся применениями $find$, которые будут также специализированы. Итеративное применение этих шагов даст код для $g$, представленный ниже.

Идея такой трансляции заключается в том, что операции композиции могут быть заменены частичными применениями функций. Это позволяет сохранять справедливость того факта, что контекстные функции не поднимаются выше первого порядка. Таким образом получается трансляция в чистые функции второго порядка.

```
1. assert(not((find_Rec(f) τ a) * 3 < 17))
2. find_Rec(f) τ x =
3.   if [[g]]_τ then find_ε τ x
4.   else find_σ∘Rec(f) τ x
5. find_ε τ x = τ x
6. find_σ∘Rec(f) τ x = find_Rec(f) (find_σ τ) x
7. find_σ τ x = …
```

**Замечание.** Есть несколько способов улучшить эту трансляцию. Во-первых, можно специализировать не только по куче, но и по типу локации, что даст более специализированные функции. Это также необходимо, чтобы получить из алгоритма трансляции типобезопасные функции. Во-вторых, полученная программа может быть частично исполнена, чтобы удалить тривиальные чтения, как, например, $\text{find}_\epsilon$. В-третьих, именованные локации могут передаваться как обычные аргументы, так как их адреса никогда не меняются. Эти три улучшения позволяют получить автоматическую трансляцию, результаты которой будут схожи с приведённой в [27] (прил. А).

## 7.3 Корректность трансляции в чистые функции

Следующая теорема (которую мы оставляем без доказательства) говорит о корректности алгоритма из разд. 7.2.

**Теорема 18.** Пусть $H \in Heap$, $\sigma_0 \in \Sigma_G$, $g$ — символьное ограничение. Тогда $\tau \bullet g$ выполнимо для некоторого $\tau \in \Sigma_G$, такого что $\sigma_0 \circ H \to^* \tau$, т. и т. т., к. небезопасна

функциональная программа, полученная из $\sigma_0 \circ H$ и $g$ применением алгоритма кодирования обобщённых куч из разд. 7.2.

Таким образом, сводя воедино корректность символьного исполнения и кодирования, получаем следующую теорему.

**Теорема 19.** Пусть $isSafe(p)$ — оракул, проверяющий безопасность функциональных программ, который всегда возвращает верный результат. Тогда программа на демонстрационном языке безопасна т. и т. т., к. алгоритм с лист. 4 выводит $Errors = \varnothing$.

*Доказательство.* Следует из теор. 17 и теор. 18.

## 7.4 Верификация функциональных программ без эффектов

Для доказательства безопасности функциональных программ без эффектов можно применить различные классические техники. Одной из самых успешных является *вывод уточнённых типов* (refinement type inference) [10, 23, 24, 25]. Фреймворки вывода уточнённых типов строят индуктивные инварианты функциональных программ высших порядков из инвариантов первого порядка над значениями с закрытыми типами (ground-types). Более точное описание этого процесса содержится в [25].

Тот факт, что получаемые в результате нашей трансляции функции не выше второго порядка, позволяет специализировать и оптимизировать процедуру вывода уточнённых типов. В контексте нашей работы, наиболее интересны *композициональные* фреймворки вывода уточнённых типов. Примером такого фреймворка является [24].

## *8. Заключение*

В работе представлен подход к композициональному точному анализу программ с динамической памятью. Подход сводит задачу доказательства корректности таких программ к задаче вывода уточняющих типов функциональных программ через построение обобщённых куч, описывающих эффект программы на произвольном состоянии. Имея хорошую модель памяти [26], подход легко адаптировать к промышленным языкам программирования, и тем самым свести задачу формальной верификации программ на таких языках к решению рекурсивно-логических соотношений. Построение практичного верификатора языка C#, основанного на представленном подходе, будет описано в следующих работах. Модель композициональной символьной памяти может также служить хорошим подспорьем для языка спецификации свойств императивных программ с динамической памятью.

Вне контекста формальной верификации работа может рассматриваться как построение интересного соответствия между императивными программами с динамической памятью и чистыми функциями: описанное в статье сведение способно по императивной программе с динамической памятью породить эквивалентную программу на чистом функциональном языке, причём порождённые функциональные программы неочевидным образом кодируют операции над динамической памятью, а композициональное сведение позволяет порождать код функций, не зависящий от её отдельных вызовов.

Предложенный в статье подход выглядит перспективным при обеспечении качества встроенных систем реального времени [28], а также при разработке операционных систем реального времени [29]. Также перспективным является интеграция данного подхода со средствами визуального моделирования ПО, в частности, при верификации исполняемых визуальных спецификаций [30].

## Список литературы / References


[1]. Distefano D. Attacking large industrial code with bi-abductive inference. Lecture Notes in Computer Science, vol. 5825, 2009, pp. 1–8.

[2]. Calcagno C. et al. Compositional shape analysis by means of bi-abduction. Journal of the ACM, vol. 58, no. 6, 2011, p. 26:1-26:66.

[3]. Gurfinkel A. et al. The SeaHorn verification framework. Lecture Notes in Computer Science, vol. 9206, 2015, pp. 343–361.

[4]. Anand S., Godefroid P., and Tillmann N. Demand-driven compositional symbolic execution. Lecture Notes in Computer Science, vol. 4963, 2008, pp. 367–381.

[5]. Distefano D. and Parkinson J M. J. jStar: Towards practical verification for Java. ACM Sigplan Notices, vol. 43, issue 10, 2008, pp. 213– 226.

[6]. Calcagno C. and Distefano D. Infer: An automatic program verifier for memory safety of C programs. Lecture Notes in Computer Science, vol. 6617, 2011, pp. 459–465.

[7]. Tillmann N. and De Halleux J. Pex–white box test generation for. net. Lecture Notes in Computer Science, vol. 4966, 2008, pp. 134– 153.

[8]. Cousot P. and Cousot R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Proc. of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, 1977, pp. 238– 252.

[9]. Khurshid S., Păsăreanu C. S., and Visser W. Generalized Symbolic Execution for Model Checking and Testing. Lecture Notes in Computer Science, vol. 2619, 2003, pp. 553–568.

[10]. Vazou N., Bakst A., and Jhala R. Bounded refinement types. ACM SIGPLAN Notices, vol. 50, issue 9, 2015, pp. 48–61.

[11]. Godefroid P. Compositional Dynamic Test Generation. ACM SIGPLAN Notices, vol. 42, issue 1, 2007, pp. 47–54.

[12]. Biere A. et al. Symbolic Model Checking without BDDs. Lecture Notes in Computer Science, vol. 1579, 1999, pp. 193–207.

[13]. Deng X., Lee J. et al. Efficient and Formal Generalized Symbolic Execution. Automated Software Engineering, vol. 19, issue 3, 2012, pp. 233–301.

[14]. Braione P., Denaro G., and Pezz`e M. JBSE: A symbolic executor for java programs with complex heap inputs. In Proc. of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016, pp. 1018–1022.

[15]. Reynolds J. C. Separation logic: A logic for shared mutable data structures. In Proc. 17th Annual IEEE Symposium on Logic in Computer Science, 2002, pp. 55–74.

[16]. Sagiv M., Reps T., and Wilhelm R. Parametric shape analysis via 3-valued logic. ACM Transactions on Programming Languages and Systems, vol. 24, issue 3. 2002, pp. 217–298.

[17]. Berdine J., Calcagno C., and O'Hearn P.W. Symbolic execution with separation logic. Lecture Notes in Computer Science, vol. 3780, 2005, pp. 52–68.

[18]. Dudka K., Peringer P., and Vojnar T. Byte-precise verification of lowlevel list manipulation. Lecture Notes in Computer Science, vol. 7935, 2013, pp. 215–237.

[19]. Barrett C. and Tinelli C. Satisfiability modulo theories. In Handbook of Model Checking, Springer, 2018, pp. 305–343.

[20]. Kahsai T. et al. Quantified heap invariants for object-oriented programs. In Proc. of the 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, 2017, pp. 368– 384.

[21]. Torlak E. and Bodik R. A lightweight symbolic virtual machine for solver-aided host languages. ACM SIGPLAN Notices, vol. 49, issue 6, 2014, pp. 530–541.

[22]. Baldoni R. et al. A survey of symbolic execution techniques. ACM Computing Surveys, vol. 51, no. 3, 2018, pp. 50:1-50:39.

[23]. Unno H., Terauchi T., and Kobayashi N. Automating relatively complete verification of higher-order functional programs. ACM SIGPLAN Notices, vol. 48, issue 1, 2013, pp. 75-86.

[24]. Zhu H. and Jagannathan S. Compositional and lightweight dependent type inference for ML. Lecture Notes in Computer Science, vol. 7737, 2013, pp. 295–314.

[25]. Cathcart Burn T., Ong C.-H.L., and Ramsay S. J. Higher-order constrained horn clauses for verification. Proceedings of the ACM on Programming Languages vol. 2, no. POPL, 2017, p. 11:1-11:27.

[26]. Мандрыкин М.У., Мутилин В.С.. Обзор подходов к моделированию памяти в инструментах статической верификации. Труды ИСПРАН, том 29, вып. 1, 2017 г., стр. 195-230 / Mandrykin M.U., Mutilin V.S. Survey of memory modeling methods in static verification tools. Trudy ISP RAN / Proc. ISP RAS, vol. 29, issue 1, 2017, pp. 195-230 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-12.

[27]. Костюков Ю. О., Батоев К. А., Мордвинов Д. А., Костицын М. П., Мисонижник А. В. Автоматическое доказательство корректности программ с динамической памятью. arXiv:1906.10204, 2019 г. / Kostyukov Yu.O., Batoev K.A., Mordvinov D.A., Kostitsyn M.P., Misonizhnik A.V.. Automatic verification of heap-manipulating programs. arXiv:1906.10204, 2019 (in Russian).

[28]. Терехов А.Н. Технология программирование встроенных систем. автореферат дис. доктора физико-математических наук. Новосибирск, 1991 г. / Terekhov A.N. Technology for programming of embedded systems. Abstract of Thesis for obtaining the degree of Doctor of physical and mathematical sciences. Novosibirsk, 1991 (in Russian).

[29]. Новиков Е.М. Развитие ядра операционной системы Linux. Труды ИСП РАН, том 29, вып. 2, 2017 г., стр. 77-96 / Novikov E.M. Evolution of the Linux kernel. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 2, 2017, pp. 77-96 (in Russian). DOI: 10.15514/ISPRAS-2017-29(2)-3.

[1]. Ольхович Л.Б., Кознов Д.В. Метод автоматической валидации UML-спецификаций на языке OCL. Программирование, том 29. № 6, 2003 г., стр. 44-50 / Ol'khovich L., Koznov D.V. OCL-based automated validation method for UML specifications. Programming and Computer Software, vol. 29, № 6, 2003, pp. 323-327.

## Информация об авторах / Information about authors

Юрий КОСТЮКОВ получил степень бакалавра в области информационных технологий в Санкт-Петербургском государственном университете в 2019 г. Его исследовательские интересы включают автоматическую верификацию, теорию моделей и конструктивную математику.

Yurii KOSTYUKOV received the bachelor’s degree in information technology from Saint Petersburg State University in 2019. His research interests include automatic verification, model theory and constructive mathematics.

Константин БАТОЕВ получил степень бакалавра в области информационных технологий в Санкт-Петербургском государственном университете в 2019 г. В число научных интересов входят анализ графов потока управления и символьное исполнение.

Konstantin BATOEV received the bachelor’s degree in information technology from Saint Petersburg State University in 2019. His research interests include control flow graph analysis and symbolic execution.

Дмитрий МОРДВИНОВ работает старшим преподавателем в Санкт-Петербургском государственном университете. В настоящее время он готовит диссертацию на соискание степени PhD в области информационных технологий. Область его научных интересов включает формальную верификацию, синтез программ и решение систем дизъюнктов Хорна.

Dmitry MORDVINOV is working as senior lecturer in Saint Petersburg State University. He is currently pursuing the Ph.D. degree in information technology. His area of interest is formal verification, program synthesis and Horn clauses solving.

Михаил КОСТИЦЫН получил степень бакалавра в области информационных технологий в Санкт-Петербургском государственном университете в 2019 г. Его исследовательские интересы включают компьютерную безопасность, анализ строковых выражений и анализ динамической памяти.

Michael KOSTITSYN received the bachelor's degree in information technology from Saint Petersburg State University in 2019. His research interests include computer security, string and heap analysis.

Александр МИСОНИЖНИК получил степень бакалавра в области информационных технологий в Санкт-Петербургском государственном университете в 2018 г. В число научных интересов входят системы типов, интуиционистская логика и теория категорий.

Aleksandr MISONIZHNIK received the bachelor's degree in information technology from Saint Petersburg State University in 2018. His research interests include type systems, intuitionistic logic and category theory.

**DOI: 10.15514/ISPRAS-2019-31(5)-4**



# Компиляция модели памяти OCaml в Power

[1,3] *Е.С. Намаконов, ORCID: 0000-0002-7517-9389 <st070466@student.spbu.ru>*
[2,3,4] *А.В. Подкопаев, ORCID: 0000-0002-9448-6587 <apodkopaev@hse.ru>*

[1] *Санкт-Петербургский государственный университет,*
*199034, Россия, Санкт-Петербург, Университетская набережная, д. 7–9*
[2] *Национальный исследовательский университет «Высшая школа экономики»,*
*194100, Россия, Санкт-Петербург, Кантемировская ул., 3А корпус 1*
[3] *JetBrains Research,*
*197342, Россия, Санкт-Петербург, Кантемировская ул., 2*
[4] *Институт им. Макса Планка: Программные Системы,*
*67663, Германия, Кайзерслаутерн, ул. Пауль-Эрлих G26.*

**Аннотация**. В настоящее время для языков программирования и процессоров активно разрабатываются модели памяти, направленные на решение различных проблем многопоточного программирования. Так, модель памяти языка OCaml (OCamlMM) позволяет избежать неопределённого поведения, вызванного гонками по данным. Для применения этой модели на практике необходимо доказать корректность её компиляции в распространённые архитектуры процессоров. На данный момент это выполнено для моделей x86 и ARM, но не для Power. Для того, чтобы упростить доказательство корректности компиляции OCamlMM в модель Power, предлагается построить схему компиляции OCamlMM в промежуточную модель памяти (IMM). Для этой модели уже доказана корректность компиляции в модель Power и другие архитектуры, поэтому доказательство корректности компиляции OCamlMM в модель Power сводится к доказательству корректности компиляции OCamlMM в IMM. В данной работе предложена схема компиляции OCamlMM в IMM и доказана её корректность. В этой схеме используются барьеры памяти и инструкции compare-and-swap, которые позволяют исключить поведение, допустимое IMM и запрещённое моделью OCaml. Полученная схема компиляции даёт корректную схему компиляции OCamlMM в модель Power. Кроме того, при таком подходе доказать корректность компиляции OCamlMM в другую архитектуру можно, доказав корректность компиляции IMM в эту архитектуру.

**Ключевые слова**: слабые модели памяти; корректность компиляции; многопоточность; OCaml memory model; IMM.

**Для цитирования**: Намаконов Е.С., Подкопаев А.В. Компиляция модели памяти OCaml в Power. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 63-78. DOI: 10.15514/ISPRAS-2019-31(5)-4

**Благодарности**: Авторы благодарят коллег по научной группе за участие в обсуждении ранних версий статьи.

# Compilation of OCaml memory model into Power

[1,3] *E.S. Namakonov, ORCID: 0000-0002-7517-9389 <st070466@student.spbu.ru>*
[2,3,4] *A.V. Podkopaev, ORCID: 0000-0002-9448-6587 <apodkopaev@hse.ru>*

[1] *St Petersburg University,*
*7–9, Universitetskaya embankment, St Petersburg, Russia, 199034.*
[2] *National Research University «Higher School of Economics»,*
*3A bld. 1, Kantemirovskaya st., St Petersburg, Russia, 194100.*
[3] *JetBrains Research,*
*2, Kantemirovskaya st., St Petersburg, Russia, 197342.*
[4] *Max Planck Institute for Software Systems,*
*G26, Paul-Ehrlich st., Kaiserslautern, Germany, 67663.*

**Abstract**. The development of memory models aimed at solving various concurrency problems is an active research topic. One such model is the OCaml memory model (OCamlMM), which allows to mitigate undefined behavior caused by data races. To use this model in practice one has to prove the correctness of its compilation into mainstream CPU architectures. At the moment, it is done for x86 and ARM but not for Power.One way to prove the correctness of compilation of OCamlMM into the Power model is to develop a compilation scheme from OCamlMM into the Intermediate Memory Model (IMM). It would be sufficient since there already exists a compilation scheme from IMM to the Power model. In this paper, the compilation scheme from OCamlMM into IMM is presented and proved to be correct. Memory fences and compare-and-swap instructions are used to permit only behavior allowed by OCamlMM. The resulting compilation scheme gives a correct compilation scheme from OCamlMM to the Power model. Moreover, when a compilation scheme from IMM into another CPU architecture will be developed, such an approach would immediately give a correct scheme of compilation of OCamlMM into that architecture.

**Keywords**: weak memory models; compilation correctness; concurrency; OCaml memory model; IMM.

**For citation**: Namakonov E. S., Podkopaev A. V. Compilation of OCaml memory model into Power. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019, pp. 63-78 (in Russian). DOI: 10.15514/ISPRAS-2019-31(5)-4

**Acknowledgements**. Authors are grateful to the research group colleagues for participating in early paper versions' discussions.

## *1. Введение*

Семантика языка программирования, поддерживающего параллельные вычисления, определяет множество возможных состояний системы (главным образом – оперативной памяти) после выполнения программы посредством *модели памяти*. Наиболее известная модель памяти – модель *последовательной согласованности* (sequential consistency, SC [1]), в которой любой результат исполнения программы может быть получен путём попеременного исполнения инструкций отдельных потоков согласно программному порядку в них. Однако из-за оптимизаций, выполняемых компилятором и процессором, могут наблюдаться поведения, невозможные в такой модели. Например, при выполнении программы[1] на рис. 1, написанной на C++ и скомпилированной с помощью компилятора GCC, на архитектуре x86 оба потока могут прочитать значение 0, что объясняется буферизацией записи (store buffering, [2] [3]).

Так как отказ от подобных оптимизаций нежелателен, современные модели памяти допускают некоторые сценарии поведения, невозможные в модели SC. Такие модели памяти называются *слабыми*. Например, слабыми являются модели памяти языков C++

[1] Здесь и далее используется упрощённый синтаксис программ: $x$ и $y$ обозначают адреса в памяти, $a$ и $b$ — локальные переменные (регистры), $rlx$ – режим доступа. В комментариях указаны наблюдаемые при чтении значения.

[4], Java [5] и JavaScript [6] [7], а также архитектур x86 [2] [8], Power [9] [10]и ARM [11] [12] [13].

| $x = y = 0$ | |
|---|---|
| $[x]^{rlx} := 1;$ | $[y]^{rlx} := 1;$ |
| $a := [y]^{rlx};$ //0 | $b := [x]^{rlx};$ //0 |

*Рис. 1. Буферизация записи*
*Fig. 1. Store buffering*

Так как отказ от подобных оптимизаций нежелателен, современные модели памяти допускают некоторые сценарии поведения, невозможные в модели SC. Такие модели памяти называются *слабыми*. Например, слабыми являются модели памяти языков C++ [4], Java [5] и JavaScript [6] [7], а также архитектур x86 [2] [8], Power [9] [10]и ARM [11] [12] [13]. Для усиления гарантий на поведение программы в слабой модели памяти необходимо использовать инструкции с более строгим *режимом доступа*: например, модель памяти C++ гарантирует, что если в программе на рис. 1 заменить режим доступа всех инструкций с $rlx$ на $sc$, то поведение полученной программы будет согласовано с моделью SC, поскольку такие инструкции будут скомпилированы с использованием т.н. барьеров памяти, запрещающих перестановку инструкций программы.

Одной из проблем таких моделей памяти является то, что они предоставляют слабые гарантии на поведение программ, содержащих гонки по данным. В частности, в модели C++ поведение таких программ не определено (см. [4], раздел 2). А модель памяти Java допускает чтение произвольных значений по отдельному адресу, если раньше по нему произошла гонка (см. [14]).

Для решения этой проблемы была предложена модель памяти OCaml (далее – `OCamlMM`) [15], обладающая свойством локальной свободы от гонок (Local Data Race Freedom property): результат обращения по данному адресу в памяти не зависит от гонок по другим адресам, а также от предыдущих гонок по этому же адресу. Это свойство гарантирует, что результат исполнения всех участков программы, не содержащих гонок по данным, будет согласовано с моделью SC.

Для того, чтобы использовать `OCamlMM` на практике, необходимо доказать её реализуемость на распространённых архитектурах процессоров. Для этого нужно доказать корректность компиляции в каждую из них – показать, что для любой программы при замене инструкций языка на процессорные инструкции согласно схеме компиляции получается программа, все сценарии поведения которой разрешены `OCamlMM` для исходной программы. В [15] приведены схемы компиляции `OCamlMM` в модели x86 и ARMv8 и доказана их корректность. При этом отсутствует схема компиляции в модель Power – архитектуру, часто используемую в серверном оборудовании [16]. Задача построения такой схемы осложнена тем, что модель Power, в отличие от моделей x86, ARMv8 и `OCamlMM`, не обладает т.н. свойством *multicopy atomicity*, т.е. не гарантирует, что записанные в память значения становятся доступны всем потокам в одном и том же порядке [11].

Для доказательства корректности компиляции `OCamlMM` в модель Power достаточно построить корректную схему компиляции `OCamlMM` в промежуточную модель памяти (Intermediate Memory Model, далее – `IMM`) [17], для которой уже доказана корректность компиляции в модель Power. Использование `IMM` как промежуточного этапа компиляции позволяет разбить доказательство корректности компиляции модели языка в модель архитектуры на два, которые впоследствии можно использовать в других доказательствах для этих моделей.

В данной работе предлагается схема компиляции OCamlMM в IMM и доказывается её корректность. Так как для IMM корректность компиляции в модель Power уже доказана, полученная схема даёт корректную схему компиляции OCamlMM в модель Power.

Статья имеет следующую структуру. Разд. 2 описывает проблему корректности компиляции OCamlMM в IMM на примерах. В разд. 3 приводится определение графа исполнения – способа представить исполнение программы в декларативных моделях памяти, к которым относятся OCamlMM и IMM. Затем, в разд. 4 описываются формальные модели OCamlMM и IMM. В разд. 5 и 6 предлагается схема компиляции OCamlMM в IMM и доказывается её корректность. В разд. 7 приводится обзор связанных работ. Наконец, в разд. 8 подводятся итоги работы и описываются направления дальнейших исследований.

## 2. Корректность компиляции OCamlMM в IMM на примерах

OCamlMM и IMM определены декларативно. Это означает, что исполнение программы представляется в виде т.н. графа исполнения. Пример программы и одного из графов её исполнения приведён на рис. 2.

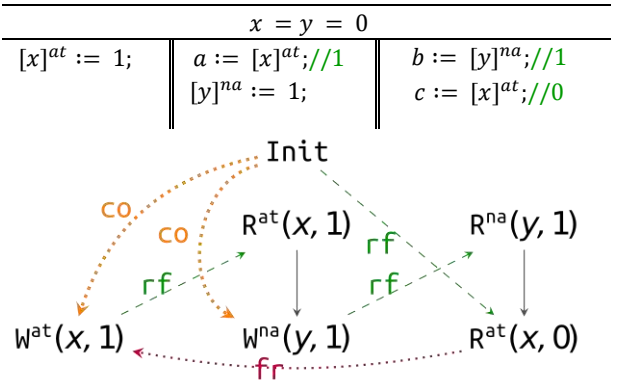


*Рис. 2. Программа и пример её исполнения, не согласованного в OCamlMM*
*Fig. 2. A program and its execution inconsistent under OCamlMM.*

Вершины графа соответствуют событиям – операциям над памятью, которые производятся при выполнении инструкций программы. Так, событие $W^{at}(x, 1)$ соответствует записи по адресу $x$ значения 1 в режиме $at$. Кроме того, в графе выделяются инициализирующие события, которые соответствуют инициализирующей записи нулей в память. На рис. 2 они для краткости объединены в множество $Init$; далее в графах мы будем опускать эти события, если это не будет важно для рассуждений.

Рёбра графа задают бинарные отношения между событиями. В данном графе есть четыре различных отношения: рёбра $po$ соответствуют программному порядку инструкций, $rf$ – чтению записанного ранее значения, $co$ – порядку записей по одному адресу, $fr$ – чтению до указанного события записи. Отношения $po$ и $co$ являются транзитивными, поэтому для их задания достаточно указывать только непосредственные рёбра. Кроме того, для краткости будем опускать подпись "$po$" рядом с соответствующими рёбрами.

*Согласованными* (допустимыми моделью) называются те исполнения, графы которых удовлетворяют некоторому предикату, заданному моделью. В частности, предикат согласованности OCamlMM требует, чтобы в графе не было циклов, состоящих только из рёбер $co$ и $fr$, проходящих между вершинами с меткой $at$, а также рёбер $po$ и $rf$. Это условие формализует свойство multicopy atomicity, описанное выше.

Граф исполнения на рис. 2 не является OCamlMM-согласованным. Действительно, это исполнение нарушает свойство multicopy atomicity: второй поток читает записанное в $x$ значение 1 до записи 1 в $y$, однако третий поток читает старое значение 0 из $x$ после чтения 1 из $y$. Соответствующий граф исполнения не удовлетворяет предикату согласованности OCamlMM, так как между вершинами есть цикл, подходящий под описание выше. Таким образом, в OCamlMM после исполнения программы на рис. 1 переменные $a$, $b$ и $c$ не могут содержать значения 1, 1 и 0 соответственно

| $x = y = 0$ | | |
|---|---|---|
| $[x]^{sc} := 1;$ | $a := [x]^{sc};//1$<br>$[y]^{rlx} := 1;$ | $b := [y]^{rlx};//1$<br>$c := [x]^{sc};//0$ |

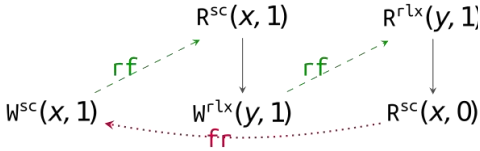


*Рис. 3. Результат компиляции программы на рис. 2 с использованием тривиальной схемы компиляции и его IMM-согласованный граф исполнения*
*Fig. 3. The result of compilation of the program from fig. 2 using the trivial compilation scheme and its IMM-consistent execution graph*

Скомпилированная программа не должна демонстрировать поведение, запрещённое исходной моделью памяти. Поэтому граф исполнения скомпилированной программы не должен быть согласованным в целевой модели памяти.

На рис. 3 показана программа, полученная в результате компиляции программы на рис. 2 согласно тривиальной схеме компиляции, и граф её исполнения. Такая схема лишь заменяет режимы инструкций на их аналоги в IMM: $na$ заменяется на $rlx$, а $at$ – на $sc$; дополнительных инструкций не вводится. Соответственно, граф на рис. 3 отличается от графа на рис. 2 только метками вершин, и в нём сохраняется цикл того же вида. IMM не гарантирует свойство multicopy atomicity, и потому предикат её согласованности не требует отсутствия таких циклов, что делает граф на рис. 3 IMM-согласованным, а соответствующее ему поведение – разрешённым. Поэтому тривиальная схема компиляции не является корректной.

На рис. 4 приведена программа, полученная в результате компиляции программы на рис. 2 согласно схеме компиляции, приведённой в разделе 5. В результате компиляции в ней появляются инструкции барьеров памяти, запрещающие некоторые оптимизации процессора и компилятора. С ними граф исполнения перестаёт быть согласованным: из рёбер $fr, po$, окружённого барьерами $rf$, а также $rf$ между событиями с меткой $sc$ образуется цикл, запрещённый в IMM.

| $x = y = 0$ | | |
|---|---|---|
| $fence^{acq};$<br>$exchg(x, 1);$ | $fence^{acq};$<br>$a := [x]^{sc};//1$<br>$fence^{acqrel};$<br>$[y]^{rlx} := 1;$ | $b := [y]^{rlx};//1$<br>$fence^{acq};$<br>$c := [x]^{sc};//0$ |

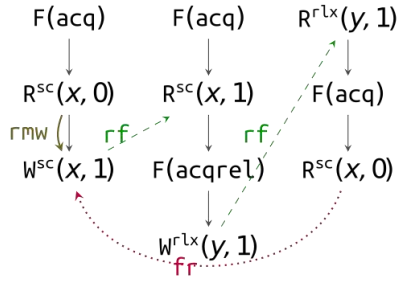


*Рис. 4. Результат компиляции программы на рис. 2 с использованием тривиальной схемы компиляции и его IMM-согласованный граф исполнения*
*Fig. 4. The result of compilation of the program from fig. 2 using the trivial compilation scheme and its IMM-consistent execution graph.*

Таким образом, для доказательства корректности компиляции необходимо доказать следующую теорему.

**Теорема** 2.1. *Пусть* $G_I$ – IMM-*согласованный граф исполнения, соответствующий графу исполнения* $G_O$ *в* OCamlMM. *Тогда* $G_O$ *является* OCamlMM-*согласованным.*

Для доказательства теоремы достаточно доказать OCamlMM-согласованность графа $G_I$. Из этого следует OCamlMM-согласованность графа $G_O$, так как он фактически является подграфом $G_I$, а условия OCamlMM-согласованности графа таковы, что они выполняются для любого его подграфа. Условие OCamlMM-согласованности состоит в иррефлексивности одного отношения и ацикличности другого. Для каждого из этих отношений доказывается включение в такое отношение, для которого в IMM-согласованном графе соответствующее условие выполняется.

## *3. Понятие графов исполнения*

В описаниях декларативных моделей памяти часто встречаются следующие обозначения отношений между вершинами. Для бинарного отношения $R$ обозначения $R^?$, $R^+, R^*$ соответствуют его рефлексивному, транзитивному и транзитивно-рефлексивному замыканиям соответственно. Обратное отношение записывается как $R^{-1}$, а области определения и значений – $dom(R)$ и $codom(R)$. Тождественное отношение на множестве $A$ обозначается как $[A]$, что часто используется в обозначениях вида $[A]; R; [B] \triangleq R \cap (A \times B)$. Левая композиция отношений записывается как $R1; R2 \triangleq \{x, y \mid \exists\, z.\, (x, z) \in R1 \wedge (z, y) \in R2\}$. Непосредственные рёбра $R$ обозначаются как $R|_{imm} \triangleq R \backslash (R; R)$.

В данном разделе описываются графы исполнения в наиболее общем виде, без привязки к конкретным моделям памяти или языкам.

Считаем, что анализируемая программа $P$ состоит из последовательных подпрограмм отдельных потоков $P_i : P = ||_{i \in Tid} P_i$, где $||$ – оператор параллельной композиции программ, $Tid$ – конечное множество идентификаторов потоков.

**Определение** 1. *Граф исполнения* $G$ задаётся множеством вершин $G.E$, отображением $G.Lab$, задающим параметры операций над памятью, и бинарными отношениями на вершинах.

Множество $G.E$ делится на инициализирующие события вида $Init\ loc$ и неинициализирующие события вида $ThreadEvent\ tid\ n$, где:

- $loc \in Loc$ – адрес инициализации, где $Loc$ – конечное множество адресов;
- $i \in Tid$ – номер потока;
- $n \in Q$ – порядковый номер внутри потока (нумерация плотным порядком упрощает формальное определение соответствия графов, см. раздел 5).

Обозначения $e.tid$ и $e.n$ для $e = ThreadEvent\ tid\ n$ соответствуют $tid$ и $n$ соответственно.

Функция $G.Lab$ сопоставляет событиям метки вида $type^{mode}(\ loc,\ val)$, где:

- $type \in \{R, W, F\}$ – тип операции (чтение, запись, барьер);
- $mode$ – один из режимов доступа, частично упорядоченных отношением “строже чем” ($\sqsupset$); конкретное множество режимов и их порядок определяется моделью памяти;
- $loc \in Loc$ – адрес памяти (для барьера не определено);
- $val \in Val$ – прочитанное/записанное значение (в случае барьера не определено), где $Val$ – множество значений, которые могут храниться в памяти.

При этом инициализирующие события обрабатываются особым образом: $G.Lab(Init\ loc) = W^{mode_{init}}(\ loc,\ val_{init})$, где $mode_{init}$ – режим доступа для

инициализирующих записей (например, в IMM $-rlx$), а $val_{init}$– начальное значение в памяти (как правило, 0).

Введём обозначения для множеств событий с определёнными метками: например, события с меткой чтения в режиме $acq$ или более строгим будем обозначать как $G.R^{acq}$ (или просто $R^{acq}$, если граф очевиден из контекста).

Рёбра графа представляют собой отношения между событиями:

- программный порядок (program order): $G.po(x,y) \Leftrightarrow (x \in Init \wedge y \notin Init) \vee (x.tid = y.tid \wedge x.n < y.n)$;
- порядок согласованности (coherence order): $G.co = \cup_{l \in Loc}\ co_l$, где $co_l$ – тотальный порядок на событиях записи по адресу $l$;
- наблюдение записанного значения (reads from, «читает-из»): $G.rf \subseteq \cup_{l \in Loc}\ G.W_l \times G.R_l$, где $G.rf(w,r) \Rightarrow G.Lab(w).val = G.Lab(r).val$, $codom(G.rf) = G.R$ и $G.rf$ является функциональным отношением;
- чтение до указанной записи: $G.fr = G.rf^{-1}; G.co$ (from-read, «читает-перед»).

В различных моделях памяти в граф исполнения могут добавляться другие отношения. Например, в IMM также есть отношение $rmw \subseteq \cup_{l \in Loc}\ [G.R_l]; po|_{imm}; [G.W_l]$, соответствующее паре событий чтения и записи в операции read-modify-write.

Введём понятие сужения графа на поток $i$: $G_i.E = \{e \in G.E | e.tid = i\}, G_i.Lab = G.Lab$.

**Определение** 2. *Графом исполнения программы* $P$ называется такой граф исполнения $G$, что его сужение на любой поток $i$ является *однопоточным графом исполнения* программы $P_i$. Соответствие подпрограммы потока и однопоточного графа исполнения определяется средствами операционной семантики, специфичной для языка ( [15], [17], [18]). Мы не приводим подробностей здесь, скажем лишь, что такая семантика задаёт соответствие между выполнением инструкций языка и изменением графа исполнения. Так, для IMM выполнение инструкции $[x]^{rel} := 1$ соответствует добавлению в текущий граф вершины с очередным порядковым номером и меткой вида $W^{rel}(x, 1)$ и рёбер, отражающих синтаксические зависимости данной записи.

**Определение** 3. *Результатом работы графа* $G$ называется функция $f : Loc \to Val$, отображающая адрес в последнее (согласно порядку $co$) записанное по нему значение. Декларативная модель памяти задаётся предикатом согласованности, которому должны удовлетворять графы исполнения программ.

**Определение** 4. *Результатом работы программы* $P$ в модели памяти $M$ является результат работы некоторого графа её исполнения, удовлетворяющего предикату согласованности $M$.

## *4. Описание используемых моделей памяти*

В данном разделе описываются рассматриваемые модели памяти и их предикаты согласованности.

### 4.1. OCaml Memory Model

OCamlMM задана в [15] эквивалентными операционным и декларативным описаниями. Для доказательства корректности компиляции будет использоваться декларативное описание.

OCamlMM поддерживает два режима доступа: неатомарный $na$ и атомарный $at$ (схожи с $pln$ и $sc$ в C++). При этом память также разделена на неатомарные и атомарные адреса, и к конкретному адресу можно обратиться только операцией соответствующего режима.

В графе исполнения OCamlMM есть только операции чтения и записи, барьеры отсутствуют.

Перед рассмотрением предиката согласованности введём ещё несколько обозначений. Для отношения $R$ будем обозначать $Ri$ рёбра $R$, проходящие между вершинами одного потока, а $Re$ –между вершинами разных потоков.

**Определение** 5. Исполнение называется `OCamlMM`-*согласованным*, если в соответствующем графе исполнения выполняются следующие аксиомы:

1) отношение $hbo$; $(co \cup fr)$ иррефлексивно, где
   $hbo \triangleq po \cup [E^{at}]; (co \cup rf); [E^{at}]$;
2) отношение $po \cup rfe \cup [E^{at}]; (coe \cup fre); [E^{at}]$ ациклично.

## 4.2. Intermediate Memory Model

Для построения схемы компиляции мы пользуемся $\texttt{IMM}_{\texttt{sc}}$ [19] – расширением `IMM`, которое дополняет оригинальную модель [17] `sc`-операциями.

`IMM` определена декларативно. Полный предикат согласованности `IMM` достаточно сложен, поэтому мы рассмотрим лишь часть модели, которая будет необходима для построения схемы компиляции.

Синтаксис программ в `IMM` напоминает таковой в C++ – помимо инструкций атомарного чтения и записи есть инструкции барьеров памяти, а также операций read-modify-write. Пары событий чтения и записи, порождаемых инструкциями read-modify-write, связаны отношением $rmw \subseteq ([G.R]; po|_{imm}; [G.W])_{loc}$. Доступно несколько режимов доступа, упорядоченных следующим образом: $\sqsubset \triangleq \{ (rlx, acq), (rlx, rel), (acq, acqrel), (rel, acqrel), (acqrel, sc)\}$.

Введём ещё несколько обозначений. $R_{loc}$ будем обозначать рёбра $R$, проходящие между вершинами с метками одного и того же адреса, $R_{\neq loc}$ – между вершинами с метками разных адресов.

**Определение** 6. Исполнение называется `IMM`-*согласованным*, если в соответствующем графе исполнения выполняются следующие аксиомы:

1) отношение $hb$; $(rf \cup co \cup fr)^+$ иррефлексивно, где
   $hb \triangleq (po \cup sw)^+$
   $sw \triangleq release; (rfi \cup po^?_{loc}; rfe); ([R^{acq}] \cup po; [F^{acq}])$
   $release \triangleq ([W^{rel}] \cup [F^{rel}]; po); rs$
   $rs \triangleq [W]; po_{loc}; [W] \cup [W]; (po^?_{loc}; rfe; rmw)^*$
2) операции read-modify-write являются атомарными: $rmw \cap (fre; coe) = \emptyset$
3) отношение $ar$ ациклично, где
   $ar \supset rfe \cup bob$
   $bob \supset [R^{acq}]; po \cup po; [F] \cup [F]; po$
4) отношение $psc_{base}$ ациклично, где
   $psc_{base} \triangleq ([E^{sc}] \cup [F^{sc}]; hb^?); scb; ([E^{sc}] \cup hb^?; [F^{sc}])$
   $scb \triangleq po \cup po_{\neq loc}; hb; po_{\neq loc} \cup hb_{loc} \cup co \cup fr$.

## *5. Схема компиляции*

Предлагаемая схема компиляции `OCamlMM` в `IMM` описана в табл. 1. За основу взята схема компиляции `OCamlMM` в модель ARM из [15].

*Табл. 1. Схема компиляции модели OCaml в промежуточную модель.*
*Table 1. Scheme of compilation of OCaml model into IMM*

| OCamlMM | IMM |
|---|---|
| $r := [x]^{na}$ | $r := [x]^{rlx}$ |
| $[x]^{na} := v$ | $fence^{acqrel}; [x]^{rlx} := v$ |
| $r := [x]^{at}$ | $fence^{acq}; r := [x]^{sc}$ |
| $[x]^{at} := v$ | $fence^{acq}; exchg(x, v)$ |

Напомним, что схема компиляции корректна, если все возможные сценарии поведения скомпилированной программы являются допустимыми сценариями поведения исходной программы согласно исходной модели памяти. В случае декларативных моделей памяти это можно переформулировать так: если граф исполнения скомпилированной программы согласован с целевой моделью памяти, то соответствующий ему граф исполнения исходной программы согласован с исходной моделью памяти.

Чтобы формализовать понятие соответствия графов, опишем, чем отличаются графы исполнения скомпилированной программы и исходной. Во-первых, все вершины исходного графа сохраняются, а новые вершины добавляются согласно схеме компиляции. Во-вторых, порядок согласованности сохраняется, так как события записи в новом графе те же, что и в исходном. В-третьих, при компиляции используются инструкции CAS, поэтому в графе появляется отношение $rmw$. Наконец, так как инструкции CAS в графе выражаются с помощью пар чтения и записи, в отношении “читает-перед” появляются новые рёбра, которые указывают на значения, наблюдаемые при исполнении CAS. Формально эти условия описываются следующим образом.

Для краткости события вида $ThreadEvent\ i\ n$ будем обозначать как $\langle i, n\rangle$.

**Определение** 7. Граф исполнения по OCamlMM $G_O$, события в потоках которого пронумерованы натуральными числами, *соответствует* графу исполнения по IMM $G_I$, если выполняются следующие условия:

1) $G_I.E = G_O.E \cup \{\langle i, n-0.5\rangle | \langle i, n\rangle \in G_O.(E \backslash R^{na})\} \cup \{\langle i, n-0.25\rangle \mid \langle i, n\rangle \in G_O.W^{at}\}$
2) $G_I.Lab = \{e \to (t, l, rename(m), v)) \mid G_O.Lab(e) = (t, l, m, v)\}$
   $\cup \{\langle i, n-0.5\rangle \to (F, -, acq, -) \mid \langle i, n\rangle \in G_O.E^{at}\}$
   $\cup \{\langle i, n-0.5\rangle \to (F, -, acqrel, -) \mid \langle i, n\rangle \in G_O.W^{na}\}$
   $\cup \{\langle i, n-0.25\rangle \to (R, l, sc, v) \mid \langle i, n\rangle \in G_O.W^{at} \wedge G_O.Lab(\langle i, n\rangle) = (W, l, at, -) \wedge v \in Val\}$,
   где $rename(na) = rlx, rename(at) = sc$
3) $G_I.rmw = \{(r, w) | w \in G_I.W^{sc} \wedge r \in G_I.R^{sc} \wedge r.tid = w.tid \wedge r.n = w.n - 0.25\}$
4) $G_I.co = G_O.co$
5) $G_I.rf \supseteq G_O.rf$ .

Видно, что нумерация событий в потоках рациональными числами позволяет добавлять в граф новые события, по программному порядку находящиеся между существующими.

Будем рассматривать граф исполнения скомпилированной IMM-программы как граф исполнения исходной OCamlMM-программы. В самом деле, при компиляции лишь добавляются новые вершины и изменяются метки у существующих. Выполнив обратное преобразование, можно получить исходный граф исполнения в OCamlMM. Поэтому для

графа исполнения в IMM можно анализировать также и его согласованность по OCamlMM, заменяя в предикате согласованности OCamlMM режимы с $na$ на $rlx$ и с $at$ на $sc$.

Теперь можно объяснить выбор данной схемы компиляции: использование инструкций compare-and-swap и барьеров накладывает на исполнение программы в IMM условия, достаточно строгие для выполнения в согласованном по IMM графе первого и второго условий согласованности по OCamlMM соответственно.

Рассмотрим несколько примеров, демонстрирующих необходимость расположения барьеров.

Рис. 5 демонстрирует, что при компиляции инструкции неатомарной записи необходимо использовать именно барьер в режиме $acqrel$, а не $rel$. Такая оптимизация может показаться разумной, т.к. $sw$ (и, следовательно, $hb$) в IMM может начинаться с $F^{rel}$. Описанное поведение должно быть запрещено, так как в графе есть запрещённый в OCamlMM цикл $po; rf; po; rf; po; [E^{sc}]; fr; [E^{sc}]$. Однако IMM разрешает такое поведение, так как после первого ребра $rf$ нет ни $R^{acq}$, ни $F^{acq}$. Если бы вместо $F^{rel}$ во втором потоке располагался $F^{acqrel}$, как это предполагает схема компиляции, то через первое ребро $rf$ прошёл бы $hb$, и результирующий граф был бы запрещён IMM.

| $x = y = 0$ | | |
|---|---|---|
| $[x]^{at} := 1;$ | $a := [y]^{na};//1$ | $b := [z]^{na};//1$ |
| $[y]^{na} := 1;$ | $[z]^{na} := 1;$ | $c := [x]^{at};//0$ |

| $x = y = 0$ | | |
|---|---|---|
| $fence^{acq};$ | $a := [y]^{rlx};//1$ | $b := [z]^{rlx};//1$ |
| $exchg(x, 1);$ | $fence^{rel};$ | $fence^{acq};$ |
| $fence^{rel};$ | $[z]^{rlx} := 1;$ | $c := [x]^{sc};//0$ |
| $[y]^{rlx} := 1;$ | | |

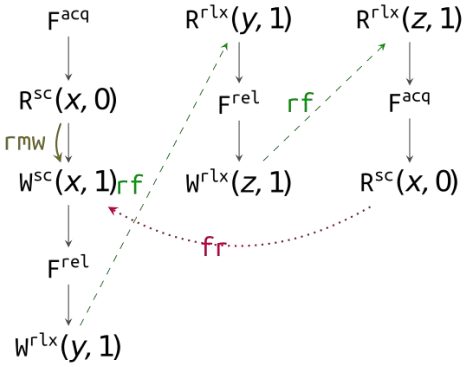


*Рис. 5. Пример программы, результата её компиляции схемой с $rel$-барьером перед инструкциями $rlx$-записи и IMM–согласованного графа её исполнения*

*Fig. 5. An example of a program, the result of its compilation with a scheme using $rel$ fence before $rlx$ write instructions and its IMM–consistent execution graph.*

Рис.6 демонстрирует, что барьер перед CAS удалить нельзя. В нём есть цикл $po; rf; po; rf; po; [E^{sc}]; fr$, запрещённый в OCamlMM, но разрешённый в IMM: sw должен оканчиваться либо $R^{acq}$, либо $R$ с последующим барьером. Ни того, ни другого во втором потоке нет, поэтому между первым и вторым потоком нельзя проложить $hb$. Если перед $R^{sc}$ во втором потоке расположить $acq$-барьер, то результирующий граф будет также запрещён и IMM.

## 6. Доказательство корректности компиляции

Напомним формулировку теоремы о корректности компиляции.

**Теорема** 2.1. Пусть $G_I$ – IMM-согласованный граф исполнения, соответствующий графу исполнения $G_O$ в OCamlMM. Тогда $G_O$ является OCamlMM-согласованным.

Как было показано выше, OCamlMM-согласованность $G_O$ следует из OCamlMM-согласованности $G_I$. Поэтому далее будем доказывать два условия OCamlMM-согласованности для $G_I$.

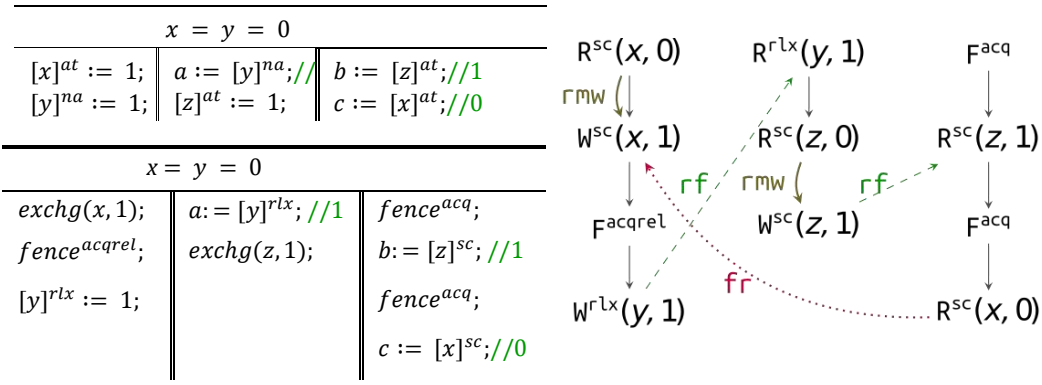


*Рис. 6. Пример программы, результата её компиляции схемой, не использующей барьер перед инструкциями $compare-and-swap$, и IMM–согласованного графа его исполнения*
*Fig. 6. An example of a program, the result of its compilation with a scheme not using a fence before $compare-and-swap$ instructions and its IMM–consistent execution graph.*

## 6.1. Первое условие OCamlIMM-согласованности

**Теорема** 6.1. Отношение $hbo; (co \cup fr)$ иррефлексивно.

Докажем, что с выбранной схемой компиляции $hbo \subseteq hb$. С учётом этого доказательство теоремы тривиально: согласно IMM-согласованности, отношение $hb; (rf \cup co \cup fr)$ иррефлексивно.

Для этого сначала покажем, что последовательность $co$ по $sc$-событиям порождает $hb$.

**Лемма** 6.2. Порядок sc-записей согласуется с отношением happens-before:
$[E^{sc}]; co; [E^{sc}] \subseteq hb$.

*Доказательство.* Заметим, что $[E^{sc}]; co; [E^{sc}]$ транзитивно. Тогда можно перейти к рассмотрению непосредственных $co$-соседей.

Предположим, что $[E^{sc}]; co|_{imm}; [E^{sc}] \subseteq [E^{sc}]; rf; [E^{sc}]; po$. Тогда доказательство тривиально: $rf$ по $sc$ событиям порождает $hb$, как и следующий за ним $po$. Значит, остаётся доказать утверждение о включении в $[E^{sc}]; rf; [E^{sc}]; po$.

Рассмотрим два события $w1, w2 \in W^{sc}$ – непосредственных $co$-соседей. По схеме компиляции перед $w2$ следуют $f \in F^{acq}$ и $r \in R^{sc}$, причём $rmw(r, w2)$.

Покажем, что $rf(w1, r)$. В самом деле, рассмотрим событие записи $w'$, из которого читает $r$. Так как обращения по одному и тому же адресу имеют один и тот же режим, то $w' \in W^{sc}$.

Пусть $w1 \neq w'$. Тогда либо $co(w', w1)$, либо, наоборот, $co(w1, w')$. В первом случае нарушается атомарность $rmw$ между $w2$ и $r$. Во втором случае получается, что между $co$-соседями $w1$ и $w2$ расположен $w'$, что невозможно. □

**Лемма** 6.3. Отношение happens-before в OCamlMM является подмножеством happens-before в IMM: $hbo \subseteq hb$.

*Доказательство.* $hbo \triangleq po \cup [E^{sc}]; (co \cup rf); [E^{sc}]$. По предыдущей лемме $co$, ограниченный на $sc$, входит в $hb$.

$rf$ по $sc$ событиями порождает $sw$, и, следовательно, $hb$. □

Таким образом, $hbo \subseteq hb$, из чего, согласно IMM-согласованности, следует, что $hbo; (co \cup fr)$ иррефлексивно. □

## 6.2. Второе условие OCamlMM-согласованности

Сначала докажем утверждения, которые позволят нам находить барьеры в программном порядке между событиями.

**Лемма** 6.4. В программном порядке между произвольным событием и записью располагается барьер: $[E \setminus F]; po; [W] \subseteq po; ([F^{acqrel}]; po; [E^{rlx}] \cup [F^{acq}]; po; [E^{sc}]); [W] \cup rmw$.

*Доказательство.* Согласно схеме компиляции, инструкция барьера располагается либо непосредственно перед инструкцией неатомарной записи, либо перед инструкцией `compare-and-swap`. Таким образом, барьера между событием и $po$-следующей записью может не быть, только если это событие чтения в $rmw$. □

**Лемма** 6.5. В программном порядке между $rlx$ и $sc$ событиями располагается барьер: $[E^{rlx}]; po; [E^{sc}] \subseteq po; [F^{acq}]; po$.

*Доказательство.* Согласно схеме компиляции, все инструкции в режиме $sc$ предваряются $acq$ барьерами. □

Кроме того, понадобятся следующие факты из алгебры:

**Лемма** 6.6. Цикл из рёбер двух типов можно представить в виде чередующихся участков рёбер каждого типа: $(x \cup y)^+ = y^+ \cup y^*; (x; y^*)^+$, где $x, y$ – произвольные отношения.

**Лемма** 6.7. Отношение $x \cup y$ ациклично, если ацикличны отношения $x, y$ и $x^+; y^+$.

Наконец, мы можем перейти ко второму условию согласованности по `OCamlMM`.

**Теорема** 6.8. Отношение $po \cup rfe \cup [E^{sc}]; (coe \cup fre); [E^{sc}]$ ациклично.

Сгруппируем первые два отношения в объединении. Тогда по лемме 6.7 нужно показать ацикличность следующих отношений:

- $po \cup rfe$;
- $[E^{sc}]; (coe \cup fre); [E^{sc}]$;
- $(po \cup rfe)^+; ([E^{sc}]; (coe \cup fre); [E^{sc}])^+$, что эквивалентно ацикличности $[E^{sc}]; (po \cup rfe)^+; [E^{sc}]; ([E^{sc}]; (coe \cup fre); [E^{sc}])^+$.

Второе и третье утверждение докажем, показав, что соответствующие отношения лежат в $([E^{sc}]; scb; [E^{sc}])^+ \subseteq psc_{base}{}^+$, где, напомним, $scb \triangleq po \cup po_{\neq loc}; hb; po_{\neq loc} \cup hb_{loc} \cup co \cup fr$ и $psc_{base} \triangleq ([E^{sc}] \cup [F^{sc}]; hb^?); scb; ([E^{sc}] \cup hb^?; [F^{sc}])$. В свою очередь, ацикличность $psc_{base}$ следует из `IMM`-согласованности графа.

Теперь видно, что второе утверждение верно по определению $scb$. По этой же причине для доказательства третьего утверждения будет достаточно показать, что $[E^{sc}]; (po \cup rfe)^+; [E^{sc}] \subseteq ([E^{sc}]; scb; [E^{sc}])^*$.

**Теорема** 6.9. Отношение $po \cup rfe$ ациклично.

*Доказательство.* Вновь воспользуемся леммой 6.7 и разложим условие ацикличности объединения на ацикличность отношений $po$ (следует из `IMM`-согласованности) и $rfe$ (двух и более таких рёбер подряд идти не может, т.к. их концы имеют разные типы), а также $po^+; rfe^+$, что эквивалентно ацикличности $po; rfe$.

Пусть такой цикл существует. Покажем, что это противоречит условию ацикличности $ar$ (что следует из `IMM`-согласованности). Напомним, что $ar \supset rfe \cup bob$ и $bob \supset [R^{acq}]; po \cup po; [F] \cup [F]; po$.

По лемме 6.4 перед событием записи, которой начинается ребро $rfe$, есть барьер $F^{\sqsupseteq acq}$, либо весь $po$ является $rmw$. В первом случае внутри $po$ есть барьер, а такое отношение лежит в $bob \subseteq ar$. Во втором случае $rmw$ начинается с $R^{sc}$, и такое ребро $po \supseteq rmw$ также содержится в $bob$. Наконец, $rfe \subseteq ar$. □

Теперь для доказательства второго условия согласованности по OcamlMM осталось доказать утверждение $[E^{sc}];(po \cup rfe)^{+};[E^{sc}] \subseteq ([E^{sc}];scb;[E^{sc}])^{*}$.

**Теорема** 6.10. Последовательность из рёбер $po$ и $rfe$ между вершинами $sc$ состоит из рёбер $scb$ между вершинами $sc$: $[E^{sc}];(po \cup rfe)^{+};[E^{sc}] \subseteq ([E^{sc}];scb;[E^{sc}])^{*}$.

*Доказательство.* Сначала введём утверждение, которое позволит отбрасывать $rfe$-рёбра.

**Лемма** 6.11. Рёбра $rfe$, у которых один из концов - $sc$, входят в $scb$: $[E^{sc}];rfe \cup [W \backslash Init];rfe;[E^{sc}] \subseteq [E^{sc}];scb;[E^{sc}]$.

*Доказательство.* Если один из концов ребра $rf$ является $sc$, таким же является и второй (исключение–чтение из инициализирующих записей, которые в IMM являются $rlx$). Тогда $[E^{sc}];rf;[E^{sc}] \subseteq [E^{sc}];hb_{loc};[E^{sc}] \subseteq [E^{sc}];scb;[E^{sc}]$. □

По лемме 6.6 имеем

$[E^{sc}];(po \cup rfe)^{+};[E^{sc}] = [E^{sc}];(rfe^{+} \cup rfe^{*};(po;rfe^{*})^{+});[E^{sc}] = [E^{sc}];(rfe \cup rfe^{?};(po;rfe^{?})^{+});[E^{sc}]$.

Начальные участки $rfe$, если они есть, можно отбросить по лемме 6.11. Рассмотрим оставшееся транзитивное замыкание:

$(po;rfe^{?})^{+} = po;(po;rfe)^{*};po^{?} = po;po^{?} \cup po;(po;rfe)^{+};po^{?} = po \cup (po;rfe)^{+};po^{?}$.

В первом случае отношение сводится к $po \subseteq scb$. Во втором, если последним ребром является $rfe$, то его можно отбросить по лемме 6.11. Остаётся случай $(po;rfe)^{+};po$. В каждой паре $po;\ rfe$ можно применить лемму 6.4. В результате нужно доказать такое утверждение:

$[(W \cup R)^{sc}];\ (po;([F^{acqrel}];po;[E^{rlx}];rfe \cup [F^{acq}];po;[E^{sc}];rfe) \cup rmw;rfe)^{+};po;[(W \cup R)^{sc}] \subseteq ([E^{sc}];scb;[E^{sc}])^{*}$.

Воспользуемся леммой 6.6: либо транзитивное замыкание состоит только из пар $rmw;rfe$, либо такие пары рёбер могут следовать после $po;([F^{acqrel}];po;[E^{rlx}];rfe \cup [F^{acq}];po;[E^{sc}];rfe)$. В первом случае замыкание имеет вид $hb_{loc} \subseteq scb$, а так как оно заканчивается $sc$-событием, оставшееся ребро $po$ также пройдёт по $sc$ и образует $scb$. Во втором случае рассмотрим, что именно находится под транзитивным замыканием:

$(po;([F^{acqrel}];po;[E^{rlx}];rfe \cup [F^{acq}];po;[E^{sc}];rfe);(rmw;rfe)^{*})^{+} =$
$((po;[F^{acq}];([F^{acqrel}];po;[E^{rlx}] \cup [F^{acq}];po;[E^{sc}]);(rfe;rmw)^{*};rfe)^{+}) =$
$((po;[F^{acq}];C;rfe))^{+}$,

где $C = C1 \cup C2 = [F^{acqrel}];po;[E^{rlx}];(rfe;rmw)^{*} \cup [F^{acq}];po;[E^{sc}];(rfe;rmw)^{*}$.

Заметим, что $(po;[F^{acq}];C;rfe)^{+} = po;[F^{acq}];(C;rfe;po;[F^{acq}])^{*};C;rfe$. В результате необходимо доказать следующее:

$[(W \cup R)^{sc}];po;[F^{acq}];(C;rfe;po;[F^{acq}])^{*};C;rfe;po;[(W \cup R)^{sc}] \subseteq ([E^{sc}];scb;[E^{sc}])^{*}$.

Заметим, что

$(C;rfe;po;[F^{acq}])^{*} = (([F^{acqrel}];po;[E^{rlx}] \cup [F^{acq}];po;[E^{sc}]);(rfe;rmw)^{*};rfe;po;[F^{acq}])^{*} \subseteq hb^{?}$, так как

$hb \triangleq (po \cup sw)^{+}$,

$sw \supset release;rfe;po;[F^{acq}]$,

$release \triangleq ([Wrel] \cup [F^{rel}];po);rs$,

$rs \supset (rfe;rmw)^{*}$.

Вспомним, что $po_{\neq loc};hb;po_{\neq loc} \subseteq scb$. Воспользуемся тем, что $po$ между между событием чтения/записи и барьером образует именно $po_{\neq loc}$. Тогда видно, что $[(W \cup R)^{sc}];po;[F^{acq}] \subseteq [E^{sc}];po_{\neq loc}$.

Остаётся показать, что $[E^{sc}];po_{\neq loc};hb;\ C;rfe;po;[(W\ \cup\ R)^{sc}] \subseteq ([E^{sc}];scb;[E^{sc}])^*$. Для этого перепишем $C\ =\ C1\ \cup\ C2$ и докажем утверждение для $C1$ и $C2$ по отдельности.

- Покажем, что $[E^{sc}];po_{\neq loc};hb^?;C1;rfe;po;[(W\ \cup\ R)^{sc}] \subseteq ([E^{sc}];scb;[E^{sc}])^*$.

Заметим, что по лемме 6.5 в последнем ребре $po$ найдётся $acq$-барьер, с помощью которого можно будет построить ребро $hb$:

$$C1;rfe;po;[(W\ \cup\ R)^{sc}] = [F^{acqrel}];po;[E^{rlx}];(rfe;rmw)^*;rfe;po;[(W\ \cup\ R)^{sc}] = [F^{acqrel}];po;[E^{rlx}];(rfe;rmw)^*;rfe;[E^{rlx}];po;[F^{acq}];po;[(W\ \cup\ R)^{sc}] \subseteq hb;po_{\neq loc};[E^{sc}].$$

Тогда

$$[E^{sc}];po_{\neq loc};hb^?;C1;rfe;po;[(W\ \cup\ R)^{sc}] \subseteq [E^{sc}];po_{\neq loc};hb^?;hb;po_{\neq loc};[E^{sc}] \subseteq [E^{sc}];scb;[E^{sc}].$$

- Покажем, что $[E^{sc}];po_{\neq loc};hb^?;C2;rfe;po;[(W\ \cup\ R)^{sc}] \subseteq ([E^{sc}];scb;[E^{sc}])^*$.

Заметим, что последовательность пар рёбер $rfe;rmw$ входит в $scb$:

$$C2\ =\ [F^{acq}];po;[E^{sc}];(rfe;rmw)^* \subseteq po_{\neq loc};[E^{sc}];([E^{sc}];rfe;[E^{sc}];rmw;[E^{sc}])^*;[E^{sc}] \subseteq po_{\neq loc};[E^{sc}];([E^{sc}];scb;[E^{sc}])^*;[E^{sc}].$$

В этом случае

$$[E^{sc}];po_{\neq loc};hb^?;C2;rfe;po;[(W\ \cup\ R)^{sc}] \subseteq [E^{sc}];po_{\neq loc};hb^?;po_{\neq loc}.$$ Тогда

$$[E^{sc}];([E^{sc}];scb;[E^{sc}])^*;[E^{sc}];rfe;po;[(W\ \cup\ R)^{sc}]\ \subseteq\ ([E^{sc}];scb;[E^{sc}]);([E^{sc}];scb;[E^{sc}])^*;([E^{sc}];scb;[E^{sc}])^2.$$ □

## 7. Связанные работы

Проблема корректности схем компиляции из `OCamlMM` и в `IMM` рассматривается и в других работах. Так, в [15] приводится схема компиляции `OCamlMM` в модель архитектуры ARMv8 [11]. В ней, в отличие от предложенной нами схемы, при компиляции неатомарной записи используется барьер $F^{acq}$, а не $F^{acqrel}$. Это объясняется тем, что в модели ARMv8 отношение $ob$ (аналог $ar$ в `IMM`) включает в себя $rfe\ \cup\ fre\ \cup\ coe$ по неатомарным операциям и $po$ с $acq$-барьером перед событием записи, поэтому в последовательности рёбер вида $(po;rfe)^+$ не требуется $rel$-барьер.

В [17] приведена схема компиляции моделей RC11 [18] в `IMM`. Так как предикат согласованности `IMM` схож с таковым в RC11, то эта схема компиляции лишь незначительно отличается от тривиальной.

В [20], [21] и [22] разработана схема компиляции модели Promising [23] в модель ARMv8 [11].

Доказательство корректности данной схемы значительно сложнее, так как модель Promising, в отличие от модели ARMv8, задана с помощью операционной семантики, что требует при доказательстве корректности компиляции задавать соответствие между графами исполнения и последовательностями шагов операционной семантики. В [17] идея этого доказательства была обобщена для построения корректной схемы компиляции `IMM` в ARMv8.

## *8. Заключение*

В данной работе представлена корректная схема компиляции `OCamlMM` в `IMM`, дающая корректную схему компиляции `OCamlMM` в модель Power. Для доказательства корректности было доказано, что в графах исполнения скомпилированных программ `IMM`-согласованность влечёт `OCamlMM`-согласованность. Так как `IMM` является более слабой моделью, чем `OCamlMM`, в предложенной схеме компиляции задействуются барьеры памяти и инструкции compare-and-swap, которые накладывают более строгие условия на поведение скомпилированной программы.

Свойство локальной свободы от гонок может быть реализовано и в других моделях памяти– например, изучается возможность включить его в модель памяти C++ [24]. Данная работа может быть использована для построения схем компиляции таких моделей.

В некоторых опубликованных доказательствах корректности компиляции впоследствии были найдены неточности. Например, [18] демонстрирует ошибку в схеме компиляции модели C++ в модель Power, [17] – в схеме компиляции модели Promising в модель Power. Чтобы избежать этого, доказательство в данной статье в дальнейшем планируется формализовать в Coq с использованием имеющейся формальной модели `IMM` [25].

## Список литературы / References


[1] Lamport L. How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Transactions on Computers, vol. C-28, isssue 9, 1979, pp. 690–691.

[2] Owens S., Sarkar S., and Sewell P. A better x86 memory model: x86-TSO. Lecture Notes in Computer Science, vol. 5674, 2009, pp. 391–407.

[3] Alglave J., Maranget L., Sarkar S., and Sewell P. Litmus: Running Tests Against Hardware. Lecture Notes in Computer Science, vol. 6605, 2011, pp. 41–44.

[4] Batty M., Owens S., Sarkar S., Sewell P., and Weber T. Mathematizing C++ concurrency. In Proc. of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 2011, pp. 55–66.

[5] Manson J., Pugh W., and Adve S.V. The Java memory model. In Proc. of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 2005, pp. 378–391.

[6] ECMA International. 2018b. ECMAScript 2018 Language Specification – Memory Model. Available at: https://www.ecma-international.

[7] Watt C., Rossberg A., Pichon-Pharabod J. Weakening WebAssembly. Proceedings of the ACM on Programming Languages, vol. 3, issue OOPSLA, 2019, 28 p.

[8] Sewell P., Sarkar S., Owens S., Nardelli F.Z., and Myreen M. O. x86-TSO: a rigorous and usable programmer's model for x86 multiprocessors. Communications of the. ACM, vol. 53, issue 7, 2010, pp. 89-97.

[9] Alglave J., Maranget L., and Tautschnig M. Herding cats: Modelling, simulation, testing, and data mining for weak memory. ACM Transactions *on* Programming Languages and *Systems*, vol, 36, issue 2, 2014, pp. 7:1–7:74.

[10] Sarkar S., Sewell P., Alglave J., Maranget L., and Williams D. Understanding POWER Multiprocessors. In Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, 2011, pp. 175-186.

[11] Pulte C., Flur S., Deacon W., French J., Sarkar S., and Sewell P. Simplifying ARM concurrency: Multicopy-atomic axiomatic and operational models for armv8. Proceedings of the ACM on Programming Languages, vol. 2, issue POPL, 2017, pp. 19:1–19:29.

[12] ARM Limited. ARM architecture reference manual: ARMv7-A and ARMv7-R edition, 2014. Available at: https://static.docs.arm.com/ddi0406/c/DDI0406C_C_arm_architecture_reference_manual.pdf.

[13] Flur S., Gray K., Pulte C., Sarkar S., Sezgin A., Maranget L, Deacon W., and Sewell P. Modelling the ARMv8 architecture, operationally: Concurrency and ISA. In Proc. of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2016, pp. 608–621.

[14] Dolan S., Sivaramakrishnan K., and Madhavapeddy A. Bounding data races in space and time. Extended version. Available at: http://kcsrk.info/papers/pldi18-memory.pdf.

[15] Dolan S., Sivaramakrishnan K., and Madhavapeddy A. Bounding data races in space and time. In Proc. of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2018, pp. 242-255.

[16] IBM Power Systems. IBM power systems facts and features: enterprise and scale-out systems with POWER8 processor technology. Available at: https://www.ibm.com/downloads/cas/JDRZDG0A.

[17] Podkopaev A., Lahav O., and Vafeiadis V. Bridging the gap between programming languages and hardware weak memory models. Proceedings of the ACM on Programming Languages, vol. 3, issue POPL, 2019, pp. 69:1–69:31.

[18] Lahav O., Vafeiadis V., Kang J., Hur C.-K., and Dreyer D. Repairing sequential consistency in C/C++11. Programming Language Design and Implementation, vol. 52, issue 6, 2017, pp. 618–632.

[19] Podkopaev A., Lahav O., Melkonian O., and Vafeiadis V. Extending Intermediate Memory Model with SC accesses. Technical report. Available at: http://plv.mpi-sws.org/imm/immsctr.pdf. 2019.

[20] Подкопаев А.В. Операционные методы в приложениях к слабым моделям памяти. Диссертация на соискание учёной степени кандидата физико-математических наук. Санкт-Петербург, 2018, 190 стр. / Podkopaev A.V. Operational methods in applications to weak memory models. The dissertation for the degree of candidate of physical and mathematical sciences. St. Petersburg, 2018,190 p. (in Russian).

[21] Подкопаев А.В., Лахав О., Вафеядис В. О корректности компиляции подмножества обещающей модели памяти в аксиоматическую модель ARMv8.3. Научно-технические ведомости СПбГПУ, том 10, № 4, 2017, стр. 51–69 / Podkopaev A.V., Lahav O., Vafeyadis V. On the correct compilation of a subset of a promising memory model into an axiomatic model ARMv8.3. St. Petersburg Polytechnic University Journal of Engineering Science and Technology, vol. 10, № 4, pp. 51-69 (in Russian).

[22] Подкопаев А.В., Лахав О., Вафеядис В. Обещающая компиляция в ARMv8.3. Труды ИСП РАН, том 29, вып. 5, 2017, стр. 149-164 / Podkopaev A.V., Lahav O., Vafeiadis V. Promising Compilation to ARMv8.3. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 5, 2017. pp. 149-164 (in Russian). DOI: 10.15514/ISPRAS-2017-29(5)-9.

[23] Kang J., Hur C.-K., Lahav O., Vafeiadis V., and Dreyer D. A promising semantics for relaxed-memory concurrency. In Proc. of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, 2017, pp. 175–189.

[24] Doherty S. Local data-race freedom and the C11 memory model. Surrey Concurrency Workshop Absracts, 2019. Available at: https://cw-srepls-19.github.io/abstracts.html#doherty/

[25] Coq formalization of the Intermediate Memory Model. Available at: https://github.com/weakmemory/imm. 2019.

## Информация об авторах / Information about authors

Егор Сергеевич НАМАКОНОВ – студент магистратуры Санкт-Петербургского государственного университета, член группы исследования слабых моделей памяти в JetBrains Research. Сферы научных интересов: слабые модели памяти, верификация ПО.

Egor Sergeevich NAMAKONOV – a Master's student in St Petersburg University, a member of weak memory model research group in JetBrains Research. Research interests: weak memory models, software verification.

Антон Викторович ПОДКОПАЕВ – доцент НИУ ВШЭ (СПб), руководитель группы исследования слабых моделей памяти в JetBrains Research, постдок Институт им. Макса Планка: Программные Системы. Сферы научных интересов: слабые модели памяти, верификация ПО, функциональное программирование.

Anton Viktorovich PODKOPAEV – an associate professor in NRU HSE (SPb), the weak memory model group leader in JetBrains Research, a postdoc in MPI-SWS. Research interests: weak memory models, software verification, functional programming.

**DOI: 10.15514/ISPRAS-2019-31(5)-5**



# Improving fuzzing performance by applying interval mutations

[1] *S.S. Sargsyan, ORCID: 0000-0002-8831-4965 <sevaksargsyan@ispras.ru>*
[1] *J.A. Hakobyan, ORCID: 0002-4094-2727 <jivan@ispras.ru>*
[1] *H.M. Movsisyan, ORCID: 0000-0002-7582-7948 <hovhannes@ispras.ru>*
[1] *M.S. Mehrabyan, ORCID: 0000-0001-9846-3414 <matos@ispras.ru>*
[1] *V.T. Sirunyan, ORCID: 0000-0002-2213-0530 <sirunyan@ispras.ru>*
[2] *Sh.F. Kurmangaleev, ORCID: 0000-0002-0558-2850 <kursh@ispras.ru>*

[1] *Russian-Armenian University,*
*123 Hovsep Emin str., Yerevan, 0051, Armenia*
[2] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
*25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*

**Abstract.** This paper presents a novel approach of generation effective inputs for fuzz testing. Most applications check input format before performing basic calculations. That kind of applications usually parse service information of input file to decide whether it is supported or not. Input formats which are not supported are discarded and the application finishes its execution immediately. For example, the service information of ELF (Extensible Linking Format) file should start with the following data: "0x7f 'E' 'L' 'F'". If a file does not contain this information in header section then it will not be considered as ELF. Effective fuzzing of an application which has input validation stage is a relevant and important problem. Random changes of input files usually malform service data and the target application finishes immediately without execution of main code. This makes fuzzing process inefficient. To solve this problem, we have designed and implemented three special plugins for ISP-Fuzzer. The first plugin is intended to collect execution traces. The second plugin connects fragments of input data and executed basic blocks of the target program. Based on that information we can determine potential fragments (critical fragments) of input data which should not be mutated for new test case generation. The third plugin is designed for interval mutations. It mutates input file escaping critical fragments detected by the second plugin. Experimental results prove the effectiveness of proposed method.

**Keywords:** dynamic analysis; interval mutation; fuzzing.

**For citation:** Sargsyan S.S., Hakobyan J.A., Movsisyan H.M., Mehrabyan M.S., Sirunyan V.T., Kurmangaleev Sh.F. Improving fuzzing performance by applying interval mutations, Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019, pp. 79-88. DOI: 10.15514/ISPRAS-2019-31(5)-5

# Повышение эффективность фаззинга с помощью интервальных мутаций

[1] *С.С. Саргсян, ORCID: 0000-0002-8831-4965 <sevaksargsyan@ispras.ru>*
[1] *Дж.А. Акоопян, ORCID: 0002-4094-2727 <jivan@ispras.ru>*
[1] *О. М. Мовсисян, ORCID: 0000-0002-7582-7948 <hovhannes@ispras.ru>*
[1] *М.С. Меграбян, ORCID: 0000-0001-9846-3414 <matos@ispras.ru>*
[1] *В.Т. Сирунян, ORCID: 0000-0002-2213-0530 <sirunyan@ispras.ru>*
[2] *Ш.Ф. Курмангалеев, ORCID: 0000-0002-0558-2850 <kursh@ispras.ru>*

[1] *Российско-Армянский Университет,*
*0051, Армения, г. Ереван, ул. Овсепа Эмина 123*
[2] *Институт системного программирования им. В.П. Иванникова РАН,*
*109004, Россия, г. Москва, ул. А. Солженицына, д. 25*

**Аннотация.** В статье представлен новый подход для генерации эффективных входных данных для фазз тестирования. Большинство программ перед началом выполнения основного кода проверяют формат входных данных. Часто такие приложения читают служебную информацию из входного файла и определяют поддерживается ли данный формат или нет. Входные файлы, с невалидным форматом отбрасываются. Эффективный фаззинг программ, которые проверяют служебную информацию входных данных является актуальной задачей. Мутация входных файлов часто приводит к генерации невалидной сервисной информации, и программа заканчивается до того, как исполнится ее основной код. Чтобы решить эту задачу, мы разработали и внедрили три специальных плагинов в платформу ISP-Fuzzer. Первый плагин предназначен для собирания трасс выполнения. Второй плагин связывает фрагменты входных данных с выполненными базовыми блоками целевой программы. С помощью этой информации определяются потенциальные интервалы входных данных, которые не должны мутироваться при генерации нового теста. Последний плагин разработан для интервальных мутаций. Эти мутации модифицируют входной файл, оставляя нетронутыми заданные интервалы. Эффективность предложенного метода доказана многочисленными экспериментами.

**Ключевые слова:** динамический анализ; интервальная мутация; фаззинг

**Для цитирования:** Саргсян С.С. Акопян Дж.А., Мовсисян О.М., Меграбян М.С., Сирунян В.Т., Курмангалеев Ш.Ф. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 79-88 (на английском языке). DOI: 10.15514/ISPRAS-2019-31(5)-5

## *1. Introduction*

Development of reliable software is still an essential aspect in the field of information technologies (IT). In order to improve software reliability, developers should repeatedly and constantly analyze and test their product. There are several methods and tools for that purpose [1-5]. Fuzzing is one of the most popular and efficient method of dynamic analysis. During analysis the target program is executed with mutated or generated input data. Fuzzing tool follows and verifies target programs behavior during its execution [6] (fig 1). If the target binary crashes then it reports about failure.

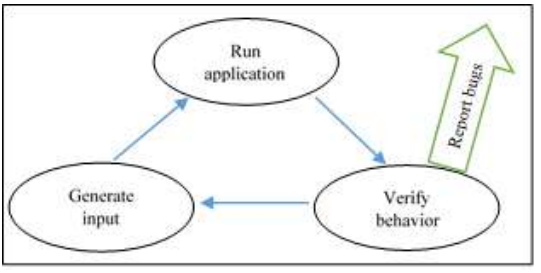


*Fig 1. Process of fuzzing*

There are number of state-of-the-art fuzzing tools which are able to detect faults in the target binary. One of them is AFL (American Fuzzy Lop) [7, 8, 9] – coverage based grey-box fuzzing tool. This tool has uncovered number of mistakes in list of widely known software such as: bash, OpenSSL and Mozilla Firefox [10]. There are several modifications of AFL for different tasks. WinAFL [11] is designed for fuzzing under Windows OS. kAFL [12] performs fuzzing of OS kernel. AFL uses an evolutionary algorithm to find new test data which will serve as input for next execution. For new input generation the tool uses a feedback loop to determine how much code was covered by current input. If the current input executes a new path then it is considered as interesting and saved for future mutation. Disadvantage of AFLs mutation engine is that it does not use any information about input file structure.

VUzzer [13] is an application aware grey box fuzzing tool which integrates static and dynamic analysis. VUzzer focuses on generation meaningful inputs which will execute new paths in target program. By static and dynamic analysis VUzzer creates a 'smart' feedback loop. Before the main fuzzing loop the tool uses static analysis to extract immediate values, magic values and other characteristic strings that affect the control flow. During the program execution, VUzzer utilize the dynamic taint analysis technique to collect information that affect the control flow branches, including specific values and the corresponding offsets. This information will be used in new test generation. VUzzer uses Pin [14], as instrumentation tool which result in a relatively slow testing speed, compared to AFL [15].

ISP-Fuzzer [16] is an extendable fuzzing framework which is implemented as coverage based, grey-box fuzzer. In order to achieve extendibility of the framework, ISP-Fuzzer provides opportunity to add custom plugins for different tasks solution. It contains number of implemented plugins, such as: BNF data generation plugin [17, 18], directed fuzzing plugin [19], DSE invocation plugin [20], etc. ISP-Fuzzer has its own mutation engine with several mutation algorithms. These algorithms, like AFL's mutation algorithms, do not know structure of input data, and may change important fragments of it such as format information in header section. For example, the first eight bytes of a PNG file contains the following values: '***0x89 0x50 0x4E 0x47 0x0D 0x0A 0x1A 0x0A***'. If fuzzing tool changes one of these values during mutation then a mutated input will be rejected by parsing stage (i.e. fuzzing tool cannot 'dig' deeper and cover new execution paths). To address this problem, we have designed and developed a plugin for ISP-Fuzzer (ZC-DSE - Zero Cost Dynamic Symbolic Execution), which detects what intervals of input data are influencing on execution of particular basic block (BB) of the target binary. Based on that information we perform interval mutations. Interval mutations can decrease possibility of changing service bytes such as: header information, magic values, etc. These mutations are implemented as ISP-Fuzzer plugin.

The rest of this paper is structured as follow. In the section 2 we present high level overview of the proposed instrument. Section 3 describes changes in instrumentation tool for proper traces generation. The section 4 describes interaction between ZC-DSE and interval mutations plugins. In the section 5 we provide results of experimental setup for number of binary files.

## *2. ZC-DSE*

ZC-DSE (Zero Cost Dynamic Symbolic Execution) is developed as a plugin for ISP-Fuzzer, to find what intervals of input data influence on execution of a particular BB of target binary. The tool accepts as input program's execution traces and corresponding input data. Algorithm which bounds intervals of input file to executed BB consist of two basic stages (each stage described in section A and B accordingly).

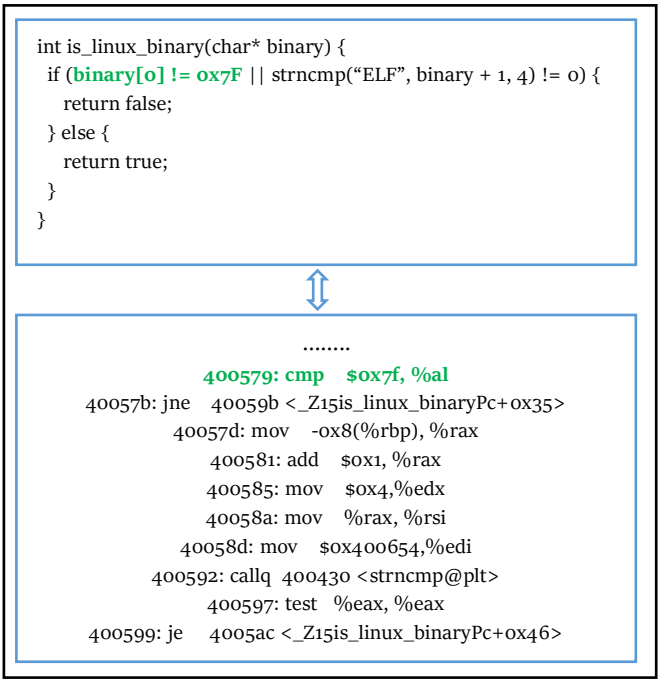


*Fig 2. Simple example of parsing header*

## 2.1 Selecting interesting basic blocks

At first the tool examines available traces to finds all interesting BBs in them. In order to consider BB as interesting, we use several metrics.

- *Most common*: BB considered as interesting if at least *P* percent of all traces contain it. Value of *P* by default is equal to 30% but can be adjusted by user. We presume, that frequently executed BBs corresponding to some service information parsing.
- *Branch:* With this metric the algorithm observes only branching BBs which have at least *T* BB on true branch and no more than *F* on the false. Values of *T* and *F* are easily tunable via configuration file. We presume that BBs satisfying these conditions are corresponding to code, which checks service information. False branch is executed when service information malformed, otherwise true branch executes basic functionality of target application.
- *Custom*: Any other metric can be easily implemented and integrated by user.

The *"Most common"* metric is used as default based on experimental results.

## 2.2 Binding interesting basic blocks to values

In the second phase ZC-DSE tries to bind intervals of input data to interesting BBs which were selected on previous stage. For that purpose, the tool intersects input files (byte level intersection) corresponding to traces containing interesting BB. We suppose resulting intervals influencing on this BB execution. If interesting BB is selected by *"Most common"* metric, and intersection of corresponding input data was empty then we stop processing of this BB. Empty result of intersection proves that there is no fixed fragment of input data influencing this BB execution.

As results ZC-DSE returns json file, which contains list of interesting BB addresses and corresponding intervals of values (from input files) which influence execution of these BBs.

For example, in fig. 2 we have fragment of code which parses header of input file and decides whether input has ELF extension. Suppose ZC-DSE have got as input two traces with respective input data (fig. 3). In these traces start address of BB2 is 0x400579 and BB2 is responsible for check whether the first position of input data is '0x7F'. In both inputs the first position is the same and equal to '0x7F'. As result ZC-DSE will return a json file which content is presented in fig. 3. ZC-DSE has determined, that in order to execute BB2 an input data should contain '0x7F' value at the first position

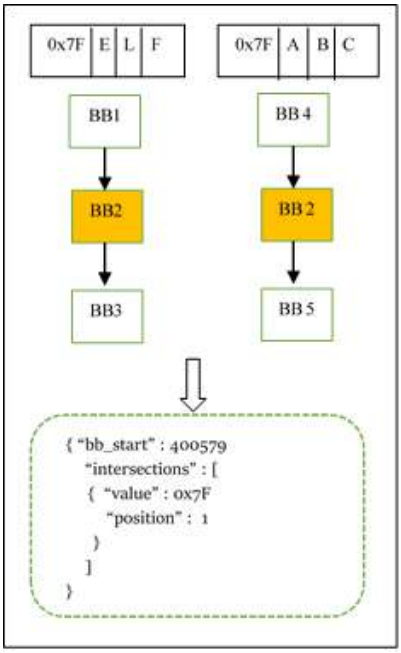


*Fig. 3. Simple example of execution ZC – DSE*

### *3. Traces generation*

ISP-Fuzzer uses DynamoRIO [21] based client library for code coverage collection. DynamoRIO is a runtime code manipulation system which allows code transformation on any part of a program, during its execution [22]. As well it provides flexible API for code manipulation.

ISP-Fuzzer supports fork server [9] optimization, which improves fuzzing speed. DynamoRIO has special code cache for instrumented BBs. It uses already instrumented BBs for next executions. Code cache and fork server optimization are strongly connected which complicates collection of current executed trace

To collect execution traces, we should dynamically collect addresses of executed BBs. For that purpose, we should dynamically inject an assembler code in each BB while its execution. The fragment will store address of current executing BB on special buffer. Injecting assembler code in all BB is expensive and influences on fuzzing speed negatively. We use special tactic to reduce trace collection cost. Every time when the fuzzing tool executes new path, we invoke instrumentation tool with special options. This option tells DynamoRIO to inject assembler code for current trace collection and run target one more time. This time current trace is collected without affecting code cache (instrumented BBs won't be stored in it). This tactic is implemented as separate plugin.

In execution trace we keep pairs of BB addresses which were executed consequentially (Fig. 4). It enables us to construct CFG (control flow graph) of current execution. Before saving a new edge into the traces buffer we check whether it already contains that edge, to optimize buffer size. Current execution traces are stored in json files (fig. 4).

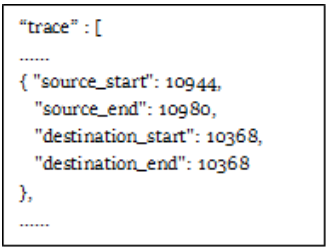


*Fig. 4. Example of execution trace fragment*

## 4. Interval mutations

ISP-Fuzzer has flexible mutation engine with number of efficient mutation algorithms. These algorithms are fast but do not use information about structure of input file. For example:

1. expand data with: same/different random byte or zero byte;
2. shrink data by removing a random number of bytes;
3. modify data by setting: sequence of bytes to random values, sequence of bytes to the same random value, random range of bytes to zero values, set a random range of bytes to random non-zero values;
4. perform a word slide;
5. flip bit/byte/word;
6. generate string data by: repeating initial string a number of times, creating a list of invalid UTF-8 strings;
7. modify string data case: lower-case each character, uppercase each character and then randomly upper-case or lowercase each character;
8. modify numeric data: set to maximum/minimum value, apply arithmetic operations;
9. insert elements from dictionary;
10. concatenate random test cases from fuzzing queue.

We have developed new mutation plugin for ISP-Fuzzer (interval mutations) which allows to let intact some parts of input data. This mutation takes as input information provided by ZC-DSE plugin and does not mutate bytes of input data, which are responsible for service information checks. By default, ISP-Fuzzer invokes ZC-DSE if original mutation algorithms do not increase coverage of target binary during 3000 executions (this number is chosen after a large number of experiments). This metric is configurable. For example, a user (analytic) can set new configuration to invoke ZC-DSE:

1. if previous *X* executions were not able to detect at least *Y* new paths;
2. if previous *X* executions were not able to execute at least *Y* new BBs.

## 5. Experimental results

We have evaluated proposed method on several binaries from Ubuntu 18.04.2 LTS. In table 1 we present some interesting difference between default mode of ISP-Fuzzer compared with ZC-DSE plugin. Results show that plugin ZC-DSE allows to detect more paths, which in its turn allows to find more crashes and hangs. All detected crashes and hangs are manually verified.

*Table 1. Results for experimental evaluation of ISP-Fuzzer with ZC-DSE*

| Program name | ISP - Fuzzer | | ISP -Fuzzer + ZC-DSE | | Difference | |
|---|---|---|---|---|---|---|
| | Found paths | Found faults | Found paths | Found faults | Paths | Faults |
| readelf | 1175 | 0 | 2081 | 1 | +906 | **+1** |
| djpeg | 48 | 0 | 45 | 1 | -3 | +1 |
| objdump | 391 | 0 | 407 | 0 | +16 | 0 |
| ar | 11 | 0 | 9 | 0 | -2 | 0 |
| latex | 80 | 0 | 196 | 1 | +116 | +1 |
| optipng | 465 | 34 | 504 | 41 | +39 | +7 |
| gif2png | 309 | 10 | 366 | 37 | +57 | +27 |
| tiff2ps | 166 | 0 | 187 | 0 | +21 | 0 |
| tiff2bw | 54 | 0 | 63 | 0 | +9 | 0 |
| tiff2pdf | 88 | 0 | 109 | 0 | +21 | 0 |
| tiff2rgba | 37 | 0 | 44 | 0 | +7 | 0 |
| jasper | 4 | 0 | 4 | 0 | 0 | 0 |
| zipclock | 91 | 0 | 108 | 1 | +17 | +1 |
| dvi2tty | 391 | 0 | 459 | 0 | +68 | 0 |
| strings | 7 | 0 | 71 | 0 | +64 | 0 |
| pdftk | 9 | 0 | 11 | 0 | +2 | 0 |

### *6. Conclusion*

Three plugins are developed in ISP-Fuzzer framework to allow interval fuzzing. The first plugin serves to collect execution traces, which will be used by the second plugin. The second plugin (ZC-DSE) is intended to bind interesting BBs of target program with values in input data. The third plugin uses information of the second one to mutate intervals of input data. Interval mutations can be used for binaries fuzzing accepting structured data. Experimental results show effeteness of the proposed method.

## References / Список литературы

[1]. V.P. Ivannikov, A.A. Belevantsev, A.E. Borodin, V.N. Ignatiev, D.M. Zhurikhin, A.I. Avetisyan. Static analyzer Svace for finding defects in a source program code. Programming and Computer Software, vol. 40, issue 5, 2014, pp 265–275.

[2]. Hayk Aslanyan, Sergey Asryan, Jivan Hakobyan, Vahagn Vardanyan, Sevak Sargsyan, Shamil Kurmangaleev. Multiplatform Static Analysis Framework for Programs Defects Detection. In Proc. of the 11th International Conference on Computer Science and Information Technologies, 2017, pp. 315-318.

[3]. H. Aslanyan, A. Avetisyan, M. Arutunian, G. Keropyan, S. Kurmangaleev and V. Vardanyan. Scalable Framework for Accurate Binary Code Comparison, In Proc. of the 2017 Ivannikov ISPRAS Open Conference, 2017, pp. 34-38.

[4]. M. Arutunian, H. Aslanyan, V. Vardanyan, V. Sirunyan, S. Kurmangaleev, and S. Gaissaryan. Analysis of Program Patches Nature and Searching for Unpatched Code Fragments. In Proc. of the 2019 Ivannikov Memorial Workshop (IVMEM), 2019, pp. 53-56.

[5]. Aslanyan H.K. Plarform for interprocedural static analysis of binary code. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018. pp. 89-100. doi: 10.15514/ISPRAS-2018- 30(5)-5.

[6]. Fuzzing (online publication). Available at: https://en.wikipedia.org/wiki/Fuzzing, accessed 11.12.2018.
[7]. American fuzzy lop (online publication). Available at: http://lcamtuf.coredump.cx/afl, accessed 11.12.2018.
[8]. American fuzzy lop for network fuzzing (unofficial) (online publication). Available at: https://github.com/jdbirdwell/afl, , accessed 11.12.2018.
[9]. Technical «whitepaper» for afl-fuzz (online publication). Available at: http://lcamtuf.coredump.cx/afl/technical_details.txt, accessed 11.12.2018.
[10]. Michał Zalewski. The bug-o-rama trophy case. Available at: http://lcamtuf.coredump.cx/afl/#bugs, accessed 11.12.2018.
[11]. WinAFL - A fork of AFL for fuzzing Windows binaries (online publication). Available at https://github.com/googleprojectzero/winafl, accessed 11.12.2018.
[12]. Schumilo, S, Aschermann C, Gawlik R, Schinzel S, Holz T. kAFL: Hardware-assisted feedback fuzzing for OS kernels. In Proc. of the 26th USENIX Security Symposium, 2017, pp. 167–182.
[13]. Rawat S., Jain V., Kumar A., Cojocar L., Giuffrida C., Bos H. Vuzzer: Application-aware evolutionary fuzzing. In Proc. of the Network and Distributed System Security Symposium, 2017, 14 p.
[14]. Luk C-K., Cohn R., Muth R., Patil H., Klauser A., Lowney G., Wallace S., Reddi V.J., Hazelwood K. Pin: building customized program analysis tools with dynamic instrumentation. ACM SIGPLAN Notices, vol. 40, issue 6, pp. 190–200.
[15]. Jun Li, Bodong Zhao, Chao Zhang. Fuzzing: a survey (online publication). Available at: https://cybersecurity.springeropen.com/articles/10.1186/s42400-018-0002-y, accessed 11.12.2018
[16]. S. Sargysan, J. Hakobyan, M. Mehrabyan, M. Mishechkin, V. Akozin, Sh. Kurmangaleev. ISP-Fuzzer: Extendable fuzzing framework. In Proc. of the 2019 Ivannikov Memorial Workshop (IVMEM), 2019, pp. 68-71
[17]. S. Sargsyan, Sh. Kurmangaleev, M. Mehrabyan, M. Mishechkin, T. Ghukasyan, S. Asryan. Grammar-based Fuzzing. In Proc. of the 2018 Ivannikov Memorial Workshop (IVMEM), 2018, pp. 32-36,.
[18]. Terence Parr. The Definitive ANTLR Reference. Pragmatic Bookshelf, 2013, 328 p.
[19]. S. Sargsyan, Sh. Kurmangaleev, J. Hakobyan, H. Movsisyan, M. Mehrabyan, S. Asryan. Directed Fuzzing Based on Program Dynamic Instrumentation. In Proc. of the 2019 International Conference on Engineering Technologies and Computer Science , 2019, pp. 30-33.
[20]. Gerasimov A.Yu., Sargsyan S.S., Kurmangaleev S.F., Hakobyan J.A., Asryan S.A., Ermakov M.K. Combining dynamic symbolic execution and fuzzing. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 6, 2018, pp. 25-38. DOI: 10.15514/ISPRAS-2018-30(6)-2.
[21]. DynamoRIO dynamic instrumentation tool platform, Feb. 2009. Available at http://dynamorio.org, accessed 11.12.2018.
[22]. Derek Bruening. Efficient, Transparent, and Comprehensive Runtime Code Manipulation. Ph.D. Thesis, MIT, September 2004.

## Информация об авторах / Information about authors

Севак Сеникович САРГСЯН – научный сотрудник, преподаватель, заведующий кафедрой, кандидат физико-математических наук. Сфера научных интересов: анализ программ, динамический анализ кода, фаззинг.

Sevak Senikovich SARGSYAN – researcher, lecturer, head of department, Ph.D in physical and mathematical sciences. Research interests: program analysis, dynamic analysis of code, fuzzing.

Дживан Андраникович АКОПЯН – научный сотрудник, преподаватель, аспирант. Сфера научных интересов: анализ программ, динамический анализ кода, фаззинг.

Jivan Andranikovich HAKOBYAN – researcher, lecturer, PhD student. Research interests: program analysis, dynamic analysis of code, fuzzing.

Оганес Мушегович МОВСИСЯН – научный сотрудник, магистр. Сфера научных интересов: анализ программ, динамический анализ кода, фаззинг.

Hovhannes Musheghovich MOVSISYAN – researcher, master. Research interests: program analysis, dynamic analysis of code, fuzzing.\ Матевос Саргисович Меграбян – научный сотрудник, магистр. Сфера научных интересов: анализ программ, динамический анализ кода, фаззинг.

Matevos Sargisovich MEHRABYAN – researcher, master. Research interests: program analysis, dynamic analysis of code, fuzzing.

Ваагн Телемакович СИРУНЯН – научный сотрудник, бакалавр. Сфера научных интересов: анализ программ, динамический анализ кода, фаззинг.

Vahagn Telemakovich SIRUNYAN – researcher, bachelor. Research interests: program analysis, dynamic analysis of code, fuzzing.

Шамиль Фаимович КУРМАНГАЛЕЕВ – старший научный сотрудник, кандидат физико-математических наук. Сфера научных интересов: анализ программ, динамический анализ кода, фаззинг.

Shamil Faimovich KURMANGALEEV – Senior researcher, Ph.D in physical and mathematical sciences. Research interests: program analysis, dynamic analysis of code, fuzzing.

DOI: 10.15514/ISPRAS-2019-31(5)-6

# Разработка языка: OOP or not OOP or better OOP

*А.Е. Недоря, ORCID: 0000-0001-8998-7072 <nedoria.aleksei@huawei.com>*

*Huawei Technologies Co., Ltd, Russian Research Institute*
*191119, Россия, Санкт-Петербург, улица Марата, 69-71*

**Аннотация.** В рамках процесса совершенствования экосистемы разработки приложений для различных устройств Huawei компания работает над новым языком программирования. В статье рассматривается подход к реализации ООР в языке программирования, который рассматривается как движение в сторону компонентно-ориентированного программирования.

**Ключевые слова:** языки программирования; компонентно-ориентированное программирование; объектно-ориентированное программирование; программная экосистема.

**Для цитирования:** Недоря А.Е. Разработка языка: OOP or not OOP or better OOP. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 89-94. DOI: 10.15514/ISPRAS-2019-31(5)-6

# Language Design: OOP or not OOP or better OOP

*A.E. Nedoria, ORCID: 0000-0001-8998-7072 <nedoria.aleksei@huawei.com>*

*Huawei Technologies Co., Ltd, Russian Research Institute,*
*69-71, Marata street, St. Petersburg, 191119, Russia*

**Abstract.** As part of the process of improving the application development ecosystem for various Huawei devices the company is working on a new programming language. The principal feature of the new language is the support of component-oriented programming (COP), by which we understand the possibility of assembling (an essential part) of the program from ready-made components. One of the steps in the direction of COP is, from our point of view, the right choice of OOP features. In the current work, we do not consider COP directly, focusing on the OO paradigm implementation. Currently, the situation with the OO paradigm is quite confusing. In fact, there is no consensus in the IT community on what OOP is. Suffice it to note that OOP in Go and Rust is fundamentally different from OOP in C++ and Java. Languages with object orientation based on classes and implementation inheritance (CLOP languages, where CLOP – Class-Oriented Porgramming) are criticized for the lack of flexibility and for the problems of developing reusable components. As component's reusing is important for us, we propose OOP features that are non-CLOP and allows one to implement objects that can be extended (by adding methods) without the need to make changes to the source code of the object and with minimal recompilation of clients.

**Keywords:** programming language; component-oriented-programming; object-oriented programming; class-oriented programming; software ecosystem.

**For citation:** Nedoria A.E. Language Design: OOP or not OOP or better OOP. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019, pp. 89-94. (in Russian). DOI: 10.15514/ISPRAS-2019-31(5)-6

## *1. Введение*

В рамках процесса совершенствования экосистемы разработки приложений для различных устройств Huawei компания работает над новым языком программирования.

Принципиальной частью экосистемы должна быть унифицированная, дружелюбная к разработчикам среда разработки, обеспечивающая разработку мульти-платформенных приложений и высокий уровень повторного использования.

Работа над языком и экосистемой идет в нескольких направлениях:

- сбор и анализ требований разработчиков;
- продумывание метрик и разработка системы измерений, позволяющей с удовлетворительной степень объективности сравнивать характеристики экосистемы и приложений, в том продуктивность разработчиков и эффективность приложений в широком смысле, включая производительность, потребление энергии, памяти и т.п.;
- разработка прототипов языка, компилятора, библиотек и среды исполнения.

Принципиальной чертой нового языка мы видим средства компонентно-ориентированного программирования (COP – Component-Oriented Programming) [1], под которым мы понимаем возможность сборки (существенной части) программы из готовых компонент. Одним из шагов в направлении COP является, с нашей точки зрения, правильный выбор средств OOP.

## *2. Подход к OOP*

В текущей работе мы не рассматриваем COP напрямую, сосредоточившись на OO парадигме. Заметим, что в настоящее время, ситуация с OO парадигмой весьма запутанная. По сути, в IT сообществе отсутствует общее понимание того, что такое OOP. Достаточно отметить, что OOP в языках Go [2, 3] и Rust [4,5] принципиально отличается от OOP в языках C++ и Java. Если добавить к списку языков Lua [6,7] и EO (Elegant Objects) [8,9], мы увидим, что словом OOP называются существенно разные и часто противоположные подходы. Для иллюстрации приведем сравнительную таблицу конструкций языков (табл. 1):

*Табл. 1. Сравнение конструкций объектно-ориентированных языков*
*Table 1. Comparison of constructions of object-oriented languages*

| **Конструкция** | **C++** | **Java** | **Go** | **Rust** | **EO** | **Lua** |
|---|---|---|---|---|---|---|
| Классы | + | + | - | - | - | - |
| Методы и атрибуты класса | + | + | - | - | - | - |
| Интерфейсы | - | + | + | + | + | - |
| Наследование абстракций или интерфейсов | + | + | - | + | + | - |
| Наследование реализации | + | + | - | - | - | - |
| Явная поддержка неизменяемости (immutability) | - | - | - | + | + | - |
| Динамическое создание классов | - | - | - | - | - | + (аналогов классов) |

Заметим, что для C++/Java наследование реализации является принципиальным и необходимым механизмом, а авторы Go, Rust и EO считают этот механизм недопустимым.

Впрочем, не будем рассуждать об OOP вообще, вспомним перечисленные выше требования к языку, которые включают продуктивность разработчика и поддержку повторного использования.

Как известно, языки, в которых объектно-ориентированность основана на классах (для этого варианта OO будем использовать аббревиатуру CLOP – Class-Oriented Programming [10]), критикуются (в том числе) за отсутствие гибкости и за проблемы разработки повторно используемых компонент [11,12,13,14,15,16]. Приведем известную цитату Джо Армстронга (Joe Armstrong), автора языка Erlang:

> *I think the lack of reusability comes in object-oriented languages, not in functional languages. Because the problem with object-oriented languages is they've got all this implicit*

> *environment that they carry around with them. You wanted a banana but what you got was a gorilla holding the banana and the entire jungle.*[1]

Источником проблемы является наследование реализации. Как мы видим, ОО языки нового поколения отказываются от классов и от наследования реализации. И, тем самым, позволяют избавиться или сделать менее острой проблему добавления горилл и джунглей к банану.

Понимая это, мы можем сделать первый шаг к определению ООР в новом языке – это **отказ от CLOP и от наследования реализаций**.

Второй шаг следует из понимания того, что возможность повторного использования компонентов **(compile once, use everywhere)** для разных устройств существенно увеличивает эффективность СОР. Частным следствием этого понимания, является **требование расширения объектов** (добавления методов) без необходимости внесения изменений в исходный код и с минимизацией перекомпиляции.

Рассмотрим пример. Определим структуру данных `List` с методами `Insert` и `Remove` (в некотором условном синтаксисе):

```
List = struct {
   fn (l: list) Append(e: Element) …
   fn (l: list) Remove(e: Element) …
}
```

Предположим, что некоторым приложения, использующим `List`, нужна операция добавления списка к списку. Рассмотрим способы реализации.

**Способ 1**. Операция реализована функцией, внешней по отношению к объекту `List`: `fn AppendList(to: list, from: list)`, использующей `to.Append` и итератор для обхода элементов `from`.

В такой реализации есть несколько недостатков:

1. функцию `AppendList` надо импортировать дополнительно (и знать о её существовании);
2. отсутствие однородности: вызов функции `AppendList` отличается от вызова `Append`;
3. Производительность отдельной функции `AppendList` скорее всего, хуже, чем метода, в котором можно использовать детали реализации.

**Способ 2**. Вписать `AppendList` в `List`. Это избавляет от перечисленных выше недостатков, но приводит к изменению исходного кода и к необходимости перекомпиляции всех программных частей, использующих `List`.

Оба этих способа нам не подходят, мы хотим обеспечить расширение, не компрометируя производительность и без необходимости перекомпиляции тех частей, которые не используют `AppendList`.

**Способ 3**. Предлагаемое решение основано на известном подходе: чтобы объединиться, надо решительно размежеваться. Выделим следующие атомарные (отдельно компилируемые) сущности:

- определение `List` как скрытого типа;
- структура данных `List:impl`, которая определяет способ реализации списка – используя массив или односвязный список или что-то третье;

[1] Перевод: Я думаю, что отсутствие возможности повторного использования свойственно объектно-ориентированным, а не функциональным языкам. Проблемой объектно-ориентированных языков является то, что с ними всегда связана неявная среда. Вы хотели банан, а получаете гориллу с бананом и все джунглями в придачу.

- отдельно каждый метод списка: `Append`, `Remove`. Для реализации каждого метода используется `List:impl`.

Теперь соберем «объект» из составных частей с помощью объединяющей конструкции, которую мы называем «`usebox`»:

```
usebox std.containers.list
export List:impl as List + Append + Remove;
```

После этого разработчик, который импортирует `std.containers.list`, может использовать `List` общепринятым способом: `var l: List; l.Append()` …

Для добавления метода `AppendList` его надо реализовать (как отдельную единицу компиляции, использующую `List:impl`), а после этого сделать новый `usebox`:

```
usebox std.containers.list2
export List:impl as List + Append + Remove + AppendList;
```

Теперь во всех программные частях, которым нужен `AppendList`, надо изменить импорт и перекомпилировать их. Те же части, которые используют только `Append` (и первый `usebox`), не требуют изменений и перекомпиляции. Кроме того, отсутствует дублирование кода, так как `usebox` – это конструкция времени компиляции. В рамках одного приложения могут использоваться `std.containers.list` и `std.containers.list2`, но код методов не будет дублироваться.

Предлагаемый механизм противоречит привычной инкапсуляции, и, на первый взгляд, уменьшает надежность программ за счет возможности доступа к внутреннему устройству `List`.

В действительности, мы просто привыкли и не замечаем странностей привычной (статической) инкапсуляции. Вообще говоря, вместо механической защиты от «всех», защищать внутреннее устройство надо от тех, кто может его испортить. Или от тех, кто не авторизован.

Для описанных атомарных сущностей можно выделить несколько уровней доступа:

- доступ к скрытому (абстрактному типу) `List` – доступно для всех разработчиков;
- доступ к коду метода – для разработчика метода;
- доступ к реализации структуры данных для использования, но не для изменения – для разработчиков методов;
- доступ к реализации структуры данных для изменения – для владельца кода (code owner).

Так как в современной разработке всегда используются системы управления кодом и версиями, технически мы готовы к переходу **к защите кода через авторизацию**.

### *3. Заключение*

Мы описали одну из черт нового языка, полезность которой сейчас проверяется с использованием прототипа компилятора и среды исполнения. Предлагаемое решение во многом опирается на предыдущие работы автора, см. [17,18,19,20,21].

## Список литературы / References

[1]. Szyperski C. Component Software: Beyond Object-Oriented Programming. Addison-Wesley Professional, 2002, 411 p.

[2]. The Go Programming Language Specification. Available at: https://golang.org/ref/spec, Version of July 31, 2019, accessed 09.10.2019.

[3]. Lukac L. Is Go an Object Oriented language? Avaliable at: https://medium.com/gophersland/gopher-vs-object-oriented-golang-4fa62b88c701, accessed 09.10.2019.

[4]. Klabnik S., Nichols C. The Rust Programming Language, Available at: https://doc.rust-lang.org/book/title-page.html, accessed 09.10.2019.

[5]. Klabnik S., Nichols C. Object Oriented Programming Features of Rust. Avaliable at: https://doc.rust-lang.org/book/ch17-00-oop.html, accessed 09.10.2019
[6]. Lua 5.3 Reference Manual. Avaliable at: https://www.lua.org/manual/5.3/, accessed 09.10.2019.
[7]. Lua. Object Orientation Tutorial. Avaliable at: http://lua-users.org/wiki/ObjectOrientationTutorial, accessed 09.10.2019.
[8]. EO, The programming language, Available at: https://github.com/yegor256/eo, accessed 09.10.2019.
[9]. Бугаенко Е. Элегантные объекты. Java Edition. Питер, Санкт-Петербург, 2019, 224 стр. / Bugaenko E. Elegant objects. Java Edition. Piter, St. Petersburg, 2019, 224 p.
[10]. Недоря А.Е. CLIP/CLOP vs pure OOP. / Nedoria A.E. CLIP/CLOP vs pure OOP. Avaliable at: http://xn--80aicaaxfgwmwf3q.xn--p1ai/?p=152, accessed 09.10.2019 (in Russian).
[11]. West D. Object Thinking. Microsoft Press, 2004, 368 p.
[12]. Suzdalnitski I. Object-Oriented Programming – The Trillion Dollar Disaster. Avaliable at: https://medium.com/better-programming/object-oriented-programming-the-trillion-dollar-disaster-92a4b666c7c7, accessed 09.10.2019.
[13]. Scalfani C. Goodbye, Object Oriented Programming. Avaliable at: https://medium.com/@cscalfani/goodbye-object-oriented-programming-a59cda4c0e53, accessed 09.10.2019.
[14]. Armstrong J. Why OO Sucks. Avaliable at: http://harmful.cat-v.org/software/OO_programming/why_oo_sucks, accessed 09.10.2019.
[15]. Will B. Object-Oriented Programming is Bad. Avaliable at: https://www.youtube.com/watch?v=QM1iUe6IofM, accessed 09.10.2019.
[16]. Church M. Was object-oriented programming a failure? Avaliable at: https://www.quora.com/Was-object-oriented-programming-a-failure/answer/Michael-O-Church?ch=10&share=cb6efe55&srid=XoXvj, accessed 09.10.2019.
[17]. Недоря А.Е. Триада языков программирования. / Nedoria A.E. The triad of programming languages. Avaliable at: http://xn--80aicaaxfgwmwf3q.xn--p1ai/?p=298, published 20.09.2018, accessed 09.10.2019 (in Russian).
[18]. Недоря А.Е. Технология разработки мультиплатформенных программ на основе явных схем программ. / Nedoria A.E. Technology for developing multi-platform programs based on explicit program schemes Available at: http://digital-economy.ru/stati/tekhnologiya-razrabotki-multiplatformennykh-programm-na-osnove-yavnykh-skhem-programm, published 04.05.2018, accessed 09.10.2019 (in Russian).
[19]. Недоря А.Е. Компонентный ассемблер для цифрового пространства. / Nedoria A.E. Component Assembler for Digital Space. Available at: http://digital-economy.ru/stati/komponentnyj-assembler-dlya-tsifrovogo-prostranstva, published 05.12.2018, accessed 09.10.2019 (in Russian).
[20]. Недоря А.Е. Компонентный ассемблер. Часть 2. Дух языка. / Nedoria A.E. Component assembler. Part 2. The spirit of language. Available at: http://digital-economy.ru/stati/komponentnyj-assembler-chast-2-dukh-yazyka, published 18.01.2019, accessed 09.10.2019 (in Russian).
[21]. Недоря А.Е. Ворчалки о программировании. / Nedoria A.E. Gruntings about programming. Available at: http://алексейнедоря.рф, accessed 09.10.2019 (in Russian).

## *Информация об авторах / Information about authors*

Алексей Евгеньевич Недоря, к.ф.-м.н., Huawei Russian Research Institute. Сфера научных интересов: языки программирования, программные экосистемы, технологии программирования, мульти-платформенное программирование, компонентно-ориентированное программирование.

Aleksei Nedoria – PhD of Physical and Mathematical Sciences, Huawei Russian Research Institute. Research interests: programming languages, software ecosystems, programming technologies, multi-platform programming, component-oriented programming.

**DOI: 10.15514/ISPRAS-2019-31(5)-7**



# Методы оценки надежности программных и технических систем

[1,2] *Е.М. Лаврищева, ORCID: 0000-0002-1160-1077 <lavr@ispras.ru>*
[1,3] *С.В. Зеленов, ORCID: 0000-0003-0446-0541 <zelenov@ispras.ru>*
[4] *Н.В.Пакулин, ORCID: 0000-0003-1266-7737<nikolay@paxdatatech.com>*

[1]*Институт системного программирования им. В.П. Иванникова РАН,*
*109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский физико-технический институт,*
*141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*
[3] *НИУ Высшая школа экономики,*
*101978, Россия, г. Москва, ул. Мясницкая, д. 20*
[4] *Pax Datatech PTE. Ltd.,*
*051531, Сингапур, Hong Lim Complex, Upper Cross Street, 531A*

**Аннотация**. Определяются основные методы обеспечения и оценки надежности и безопасности программно-технических систем в процессах их жизненного цикла, а также сбора сведений о возникающих в системах ошибках, дефектах и отказах для последующих изменений. Рассматривается стандартная модель надежности и дается характеристика базовых показателей, среди которых присутствует показатель надежности; функциональность и безопасность составляют основу измерения надежности. Приводится классификация моделей надежности, дается характеристика моделей оценочного типов, используемых при проверке показателей надежности компонентов программно-технических систем. Обсуждаются экспериментальные результаты применения оценочных моделей надежности к разным размерам программных компонентов программно-технических систем и приводится оценка результатов измерения показателя надежности на этих компонентах с учетом плотности дефектов, интенсивности отказов и восстанавливаемости. Отмечается важность обеспечения надежности и безопасности (dependability and safety) систем в рамках новых стандартов интеллектуальных систем и Интернета вещей.

**Ключевые слова:** надежность; ошибка; дефект; отказ; безопасность; тестирование; надежность; риск; умные компьютеры.

**Для цитирования:** Лаврищева Е.М., Зеленов С.В., Пакулин Н.В. Методы оценки надежности программных и технических систем**.** Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 95-108. DOI: 10.15514/ISPRAS-2019-31(5)-7

**Благодарности.** Работа поддержана грантом РФФИ 19-01-00206.

# Methods for assessing the reliability of software and hardware systems

[1,2] *E.M. Lavrischeva, ORCID: 0000-0002-1160-1077 <lavr@ispras.ru>*
[1,3] *S.V. Zelenov, ORCID: 0000-0003-0446-0541 <zelenov@ispras.ru>*
[4] *N.V. Pakulin, ORCID: 0000-0003-1266-7737<nikolay@paxdatatech.com>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*
[2] *Moscow Institute of Physics and Technology (State University), 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia*
[3] *National Research University, Higher School of Economics 20, Myasnitskaya Ulitsa,Moscow, 101978, Russia*
[4] *Pax Datatech PTE. Ltd., 531A Upper Cross Street, Hong Lim Complex, Singapore, 051531*

**Abstract.** At first, the paper introduces the main methods for ensuring and assessing the reliability and safety of software and hardware systems in the processes of their life cycle, as well as collecting information about errors that occur in systems, defects and failures for subsequent changes. Then we consider a standard reliability model and provide a characteristic of basic indicators that include a reliability indicator; functionality and safety form the basis for measuring reliability. After this, the paper demonstrates the classification of reliability models, and shows the characteristics of the evaluation types used in the verification of reliability indicators of components of software and hardware systems. Next, we discuss the experimental results of applying the estimated reliability models to different sizes of software components of software and hardware systems as well as the results of measuring the reliability indicator on these components taking into account the density of defects, failure rates, and recovery. Finally, we note the importance of ensuring the reliability and safety (dependability and safety) of systems in the framework of the new standards of intelligent systems and the Internet of things.

**Keywords:** reliability; error; defect; failure; security; fault; completeness; testing; reliability; risks; smart computers

**For citation:** Lavrischeva E.M., Zelenov S.V., Pakulin N.V. Methods for assessing the reliability of applied systems. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019, pp. 95-108 (in Russian). DOI: 10.15514/ISPRAS-2019-31(5)-7

**Acknowledgements.** This work was supported by RFBR, project 19-01-00206.

## *1. Введение*

Надежность систем – это теоретическая и прикладная наука, способствующая созданию надежных и безопасных систем и проведению измерения отдельных показателей качества (функциональность, надежность, безопасность и др.) технических и программных систем (ПТС). Методы надежности позволяют прогнозировать, измерять и оценивать качество продукта, доводя ошибки, дефекты и интенсивность отказов в компонентах систем к минимуму [1-9].

В проблеме обеспечения надёжности ПТС имеется некоторые общие положения – это возникновение случайных явлений и способы их анализа с применением теории вероятности. К системам реального времени, включая радарные системы, системы безопасности, медицинские, производственные системы и оборудование, предъявляются высокие требования к надежности (недопустимость ошибок, достоверность, защищенность и др.).

Надежность систем зависит от числа оставшихся и не устраненных ошибок. Чем интенсивнее проводится эксплуатация, тем интенсивнее выявляются ошибки и быстрее растет надежность системы [1-14]. В проблеме обеспечения надёжности ПТС лежат

случайные явления (аварии, киберугрозы, пожары и др.), которые требуют принятия серьезных мер по прогнозированию рисков и мер противодействия возникающим угрозам.

Согласно ФЗ «О безопасности», ГОСТ Р 22.0.02 и недопустимости риска ГОСТ Р 51898-2002, ГОСТ 1.1-2002 и др. в области системной инженерии требуется обеспечение качественных систем; функциональных и программно-технических интерфейсов разных видов систем; безопасности и защищенности общественных и государственных систем от внутренних и внешних угроз; работоспособности и конкурентности компьютерных систем с прогнозированием рисков и принятием сбалансированных мер, упреждающих противодействиям авариям, сбоям, киберугроз и др. [15-20].

Одним из важных условий безопасной работоспособности компьютерных систем является надежность, которая зависит от оставшихся и не устраненных ошибок в отдельных ее компонентах. Чем интенсивнее проводится эксплуатация, тем интенсивнее выявляются ошибки и быстрее растет надежность системы.

Для оценки надежности систем используются собранные статистические данные о вероятности и времени безотказной работы; отказы и частота (интенсивность) отказов, защита от искажений программ и данных, прогнозирования характеристик надежности на процессах жизненного цикла (ЖЦ) с учетом функциональности и безопасности.

В ГОСТ 27.002-2015 надежность определяется как свойство объекта сохранять во времени в установленных пределах значения всех параметров, характеризующих способность выполнять требуемые функции в заданных режимах и условиях применения, технического обслуживания, хранения и транспортирования.

Главными источниками информации об ошибочных ситуациях для оценки надежности являются процессы ЖЦ (задания требований, проектирование, тестирование, эксплуатация и испытание), например, стандарты ЖЦ (ISO/IEC 15846-1998, 15939:2002 и др.) [17-20].

На всех процессах ЖЦ могут возникать случайные ситуации:

- *отказ ПС (failure)* – это переход системы из рабочего состояния в нерабочее;
- *дефект (fault)* – это последствие выполнения элемента программы, приводящее к некоторому событию, например, в результате неверной интерпретации его компьютером или человеком; дефекты в программе, не выявленные в результате проверок, является источником потенциальных ошибок и отказов системы;
- *ошибка (error)* как следствие недостатка в описании одной из программ или при принятии неверных решений.

На процессах ЖЦ набирается статистика отказов, их интенсивность и обнаруживаются дефекты в компонентах системы. Проявляющиеся дефекты требуют создания средств защиты от случайных искажений программ и данных. Они оцениваются после проведения испытания системы и влияют на получение количественного коэффициента надежности системы [10-17].

Надежность функционирования системы характеризуется устойчивостью, восстанавливаемостью и работоспособностью системы. Непредусмотренные при проектировании ошибки и дефекты влияют на безопасность функционирования системы. Кроме того, возникают угрозы (отказы, катастрофические явления), которые влияют на функциональную безопасности системы и показатели надежности системы.

При обеспечении безопасности системы учитываются только те отказы, которые могут привести к катастрофическим последствиям и ущербам (например, пожар, взрыв, разрушение зданий и др.). Оценка надежности и безопасности функционирования системы от метрик стандарта качества (внешние, внутренние, эксплуатационные). Они сравниваются с требованиями на систему и используются при сертификации готовой системы.

Для оценки надежности и функциональной безопасности следует использовать стандарт ISO 15408 - 2008, IEC 61508, согласно которым требуется проведение защиты аппаратуры от всех видов угроз и устранение возникающих неисправностей при дефектах, сбоях и отказах.

## *2. Методы обеспечения безопасного и надежного функционирования систем*

Процесс возникновения ошибок и отказов в ПТС определяется временем их возникновения или частотой, числом и их интенсивностью. В связи с этим поиск случайных величин осуществляется стохастическими методами или вероятностными. Если случайные величины распределены по показательному, эрланговскому или гиперэрланговскому законам, то поведение системы описывается Марковским процессом.

В основном модели оценки надежности основываются на статистике отказов и распределении интенсивности выявленных отказов в ПТС. Некоторые модели надежности исходят из предположения, что найденные дефекты устраняются немедленно (или временем их устранения можно пренебречь) и при этом новые дефекты не вносятся. При этом количество дефектов в системе уменьшается, а надежность возрастает и такие модели получили название *моделей роста надежности.*

К наиболее распространенным методам обеспечения безотказной и безопасной работы систем относятся модели Джелинского-Моранды, Нельсона, Мусы, Вейса и др. [1-3]. Сформировалась классификация моделей надежности: прогнозирующие, измерительные и аналитические.

Большинство моделей оценки надежности базируются на дефектах, статистике отказов и распределении их интенсивности в ПС. Найденные дефекты устраняются немедленно и новые дефекты не вносятся. В результате количество дефектов в ПС уменьшается, а надежность возрастает. Такие модели получили название моделей роста надежности.

Известна следующая классификация моделей надежности: прогнозирующие, измерительные и оценочные.

*Прогнозирующие модели* надежности основаны на измерении технических характеристик создаваемой программы: длина, сложность, число циклов и степень их вложенности, количество ошибок на страницу операторов программы и др. (например, модель Мотли–Брукса).

*Измерительные модели* предназначены для измерения надежности ПО, работающего с заданной внешней средой и следующими ограничениями: при измерении свойств надежности ПС не изменяется; обнаруженные ошибки не исправляются; оценка надежности проводится для зафиксированной конфигурации ПО. Примером таких моделей является модель Нельсона и Рамамурти–Бастани и др.

*Оценочные модели* основываются на серии тестовых прогонов и проводятся на ЖЦ тестирования ПС. Эти модели могут быть без подсчета ошибок позволяют спрогнозировать количество ошибок, оставшихся в программе. К таким моделям относятся модели Джелински и Моранды, Шика Вулвертона и Литвуда–Вералла. К моделям с подсчетом отказов на заданных интервалах времени относятся: модели Шика–Вулвертона, Шумана, Пуассоновская модель и др.

При внесении изменений в программу проводится повторное тестирование и оценка надежности. Этот подход базируется на тестировании и генерации множества тестовых выборок из входного распределения (например, модель Нельсона и др.)

По фактору распределения интенсивности отказов модели надежности подразделяются на экспоненциальные, логарифмические, геометрические, байесовские и др. В них проблема

надежности ПО рассматривается как способность системы быть работоспособной (dependability) с атрибутами (свойствами):

- availability – готовность к использованию;
- reliability – готовность к непрерывному функционированию;
- safety – безопасность для окружающей среды (для внешнего окружения). Способность не вызывать катастрафических последствий в случае отказа;
- confidentability – секретность, сохранение секретности информации;
- integrity – способность к сохранению информации, устойчивость к её самопроизвольному изменению;
- aintainability – эксплуатационные способности ПО, простота выполнения операций обслуживания (например, устранение ошибок, восстановление после ошибки и т.п.);
- security – готовность, сохранность и скрытность (confidentability);
- failure – отказ, отклонение поведения системы от предписанного, т.е. когда система перестаёт выполнять предписанные ей функции;
- error – ошибка, состояние системы, которое вызывает отказ (mistake);
- fault – отказ, в случае причины ошибки, что вызывает её;

Достижение работоспособности обеспечивается многими методами, которые включают:

- fault preventation – предотвращение отказа,
- remival fault – устранение отказа,
- fault tolerance – возможность выполнения ПО при наличии ошибки,
- fault forecasting – возможность появления отказа и его последствия для оценки.

Различие между fault и failure не критическое и поэтому используется термин defect может означать либо fault (причина), либо failure (действие).

Таким образом, оценка надежности системы осуществляется по тем моделям надежности, которые соответствуют типу анализируемой системы. Если обнаружены ошибки и внесены необходимые изменения в нее, проводятся такие действия:

- протоколирование отказов в ходе функционирования ПС и измерение надежности функционирования, а также использование результатов измерений при определении потерь надежности в период времени эксплуатации;
- анализ частоты и серьезности отказов для определения порядка устранения соответствующих ошибок;
- оценка влияния функционирования ПС на надежность в условиях усовершенствования технологии или использования новых инструментов разработки ПС.

Таким образом, показано, что надежность является одной и главных характеристик современных ПТС, для которой разработано большое количество моделей для разных ее видов и типов. Рассмотрены основные базовые понятия надежности, обеспечивающие оценку надежности по соответствующим моделям надежности ПТС, основанным на времени функционирования и/или количестве отказов (ошибок), полученных в программах в процессе их тестирования или эксплуатации.

## *3. Эксперименты по применению моделей надежности.*

Были проведены эксперименты для моделей *оценочного* типа, которые базируются на пуассоновских процессах (модели Мусы, Гоэла–Окомото, S-образные и др.). Эксперименты проводились для небольших (1), средних (2), больших (3) и очень больших (4) проектов [2-5, 15].

*Таблица 1. Характеристики моделей надежности Пуассоновского типа для вычисления интенсивности и количества отказов*
*Table 1. Characteristics of Poisson type reliability models for calculating the intensity and number of failures*

| Название модели | Функции интенсивности отказов $\lambda(t)$ | Функция количества отказов $\mu(t)$ |
|---|---|---|
| Модель Гоэла-Окумото | $\lambda(t) = Nb\,exp(-bt)$ | $\mu(t) = N(1 - exp(-bt))$ |
| Модель Мусы | $\lambda(t) = \beta_0\,\beta_1\,exp(-\beta_1 t)$ | $\mu(t) = \beta_0(1 - exp(-\beta_1 t)$ |
| S –подобная модель | $\lambda(t) = \alpha\,\beta^2 t\,exp(-\beta t)$ | $\mu(t) = \alpha\{1 - (1 + \beta t)exp(-\beta t)\}$ |
| Модель Шнайдевинда | $\lambda(t) = \alpha_0\,exp(-\beta t)$ | $\mu(t) = \alpha_0/\beta(1 - exp(-\beta t))$ |
| Общая модель пуассоновского процесса | $\lambda(t) = \alpha\,\beta^{n+1}t^n\,exp(-\beta t)$ | $\mu(t) = \alpha\left(n^1 - \sum b\beta^{n-1}/(n-1)^1 t^n exp(-\beta t)\right)$ |

В табл. 1 представлены практически полученные значения функций интенсивности отказов $\lambda(t)$ и количество отказов $\mu(t)$ для проектов (1-4) с помощью названных в табл. 1 моделей надежности. В них значения $\alpha$ и $\beta$ находятся в следующих соотношениях: $N = \alpha, \beta = \alpha, b = \beta, \beta_1 = \beta, \alpha_0 = \alpha\beta$. Параметр *n* зависит от процесса тестирования и его значений:

- *n*=0 для небольшого проекта, в котором разработчик является также тестером (модели Мусы, Гоэло-Окомото и др.);
- *n*=1 для среднего проекта, в котором тестирование и проектирование ПО исполняются несколькими разработчиками из одной рабочей группы (S- образная модель);
- *n*=2 для большого проекта, в котором группы разработчиков работают параллельно;
- *n*=3 для очень большого проекта, в котором группы тестирования и разработки работают независимо друг от друга.

На основе проведенных экспериментальных данных получены функции о количестве отказов $\mu(t)$ и интенсивности отказов $\lambda(t)$ на выходных данных и значениях параметра *n*, который показывает вид функций $\mu(t)$ при разных значений $n$ = 0, 1, 2, 3.

Наибольшее приближение достигается при *n*=3, а наименьшее при $n$ = 0 (модель Мусы, Гоэло-Окомото и др.). По этим моделям надежность стремиться к 1. Одним из недостатков является форма кривой интенсивности выявленных отказов или неисправностей (экспоненциальная) и строго спускается при $t > 0$. Это свидетельствует о том, что при тестировании проведено недостаточно экспериментов или мало найдено ошибок, когда интенсивность отказов была близка 0. В системе могут оставаться ошибки и их поиск требует больше времени.

При применении S-образной модели, функция интенсивности $\lambda(t)$ выявления ошибок в зависимости от времени работы имеет вид: $\lambda(t) = \alpha\,\beta^2 t\,exp(-\beta t)$, где $\alpha$ – общее количество дефектов, обнаруженных от начала и до конца тестирования; $\beta$ – скорость изменения функции интенсивности при выявлении отказов.

*Таблица 2. Статистические данные для μ(t) при n=3, 2, 1 и данных t_2*
*Table 2. Statistics for μ(t) with n=3, 2, 1 and t_2 data*

| Статистические показатели отклонений | Разница функций *t_2 - μ_3* | Разница функций *t_2 - μ_2* | Разница функций *t_2 - μ_1* | Разница функций *t_2 - μ* |
|---|---|---|---|---|
| Среднее | 16.13522 | 16.22889 | 19.88387 | 58.93807 |

| Статистические показатели отклонений | Разница функций $t\_2 - \mu\_3$ | Разница функций $t\_2 - \mu\_2$ | Разница функций $t\_2 - \mu\_1$ | Разница функций $t\_2 - \mu$ |
|---|---|---|---|---|
| Медианное | 15.27700 | 14.11600 | 16.0000 | 60.89700 |
| Максимум | 33.58100 | 54.23600 | 49.10800 | 88.80200 |
| Минимум | 4. 848000 | -1.280000 | 41.75000 | 15.96200 |
| Среднеквадратическое отклонение | 8.374089 | 17.37143 | 14.07056 | 23.63765 |

Введение в формулу параметра в степени 1 модели Мусы и Гоэла–Окомото дает изменение формы кривой так, что она сначала растет, а потом спадает. Практика применения этих моделей в ПТС привела к уточнению функции интенсивности при введении дополнительного параметра n: $\lambda(t) = \alpha\beta^{n+1}t^n exp(-\beta t)$, где $n$ отражает сложность и размер проекта системы. Это позволяет более точно определить форму кривой интенсивности с учетом получаемых практических результатов.

Результат экспериментов подтверждается соответствующими статистическими данными (табл. 2), которые задают разницу между выходными данными $(t\_2)$ и соответствующими значениями функции $\mu(t)$ при значениях n = 0, 1, 2, 3. На основе экспериментальных данных $\alpha$, β, n, приведены значения функций $\mu(t)$ и $\lambda(t)$ при n = 3, 2, 1, полученные при использовании методов оценки надежности Мусы, Мусы-Окомото и Шнайдевинда.

## *4. Процессы ЖЦ для разработки компонентов систем*

Многие современные системы, работающие в реальном времени (авиа, космические, и др.) требуют высокой надежности (недопустимость ошибок, отказов, аварий и др.), которая в значительной степени зависит от оставшихся и не устраненных ошибках в процессе разработки ПТС.

На надежность ПО также влияют угрозы, приводящие к неблагоприятным последствиям, риску нарушения безопасности и способности компонентов системы сохранять устойчивость в процессе ее эксплуатации. При этом риск уменьшает надежность и безопасность системы, если проявляются внешние угрозы [4-7, 20-27].

В связи со сказанным особую важность приобретает задача применения ЖЦ при создании качественных компонентов, включая процессы:

- системный анализ и разработка требований к системе и характеристик качества функционирования;
- проектирование архитектуры (модели) системы и отдельных компонентов;
- реализация компонентов и их конфигурация в систему;
- тестирование компонентов и системы из компонентов;
- испытание системы;
- сопровождение системы.

*На процессе анализа и разработки требований* определяются функции системы и спецификация основных требований к системе с заданием метрик для оценки надежности [4, 19], в терминах интенсивности отказов или вероятности безотказного функционирования. Разработчики системы формируют:

- приоритеты функций системы по критерию важности их реализации;
- параметры среды функционирования и интенсивности выполнения функций и их отказов;
- входные и выходные данные для каждого функционального компонента системы;
- категории отказов и их интенсивность при выполнении функций в заданное время.

*На процессе проектирования* определяются:

- размеры информационной и алгоритмической сложности всех типов проектируемых компонентов;
- виды дефектов, свойственные всем типам компонентов системы;
- стратегии функционального тестирования компонентов по принципу «черного ящика» с помощью тестов для выявления дефектов и интеграционного тестирования.

Для достижения надежного продукта проводится анализ:

- вариантов архитектуры системы на соответствие требованиям к надежности;
- анализ рисков, отказов и деревьев ошибок для критических компонентов с целью обеспечения отказоустойчивости и восстанавливаемости системы;
- прогнозирование показателей размера системы, чувствительности к ошибкам, степени тестируемости, оценки риска и сложности системы;
- прогнозирование количества и плотности дефектов для проведения процесса измерения надежности.

*На процессах реализации и тестирования системы* проектные спецификации функций компонентов системы переводятся в коды и подготавливаются наборы тестов для автономного и комплексного тестирования компонентов и систему. При проведении автономного тестирования обеспечение надежности состоит в предупреждении появления дефектов в компонентах и создании эффективных методов защиты от них. Все последующие процессы разработки (например, верификация) не могут обеспечить надежность систем, а лишь способствуют повышению уровня надежности за счет обнаружения ошибок с помощью тестов различных категорий и их исправления.

*На процессе испытаний* проводится системное тестирование для установления соответствия внешних спецификаций функций целям системы. Испытание проводится в реальной среде функционирования или на испытательном стенде с помощью наборов данных для имитации функций компонентов. При подготовке к испытаниям изучается «история» тестирования (таблица сведений об ошибках и отказах) на процессах ЖЦ, использование ранее разработанных тестов для составления специальных тестов испытаний с целью [2]:

- управления ростом надежности при неоднократном исправлении и регрессионном тестировании ПТС;
- принятия решения о степени готовности системы и возможности ее передачи в эксплуатацию;
- оценки надежности по результатам системного тестирования и испытаний по соответствующим *моделям надежности*, подходящих для заданных целей системы.

*На процессе сопровождения* оценка надежности ПС проводится путем:

- протоколирования отказов в ходе работы системы, измерения надежности функционирования и использования результатов измерений для определения потерь (снижения) надежности в период времени эксплуатации;
- анализа частоты и серьезности отказов для определения порядка их устранения;
- оценки влияния функционирования системы на надежность в условиях усовершенствования технологии и применения новых инструментов разработки и тестирования.

На этапах жизненного цикла измерения надежности выполняются и решаются следующие задачи [5, 6]: определение функции распределения надежности по компонентам, прогнозирование плотности дефектов, прогнозирование надежности и оценки надежности. Рассмотрим эти задачи.

**Задача распределения надежности по компонентам** на процессах ЖЦ состоит в парном сравнения компонентов системы и построении квадратной матрицы $A(n\times n)$ из элементов вида:

$$a_{11} = a_{22} = \cdots = a_{nn} = 1, a_{ij} = \frac{1}{a_{ji}}, i, j = 1, \dots, n, i \neq j;\ n = k, l, m,$$

где $n$ – количество сравниваемых компонентов, $k$, $l$, $m$ – количество функций и модулей соответственно. Матрица включает относительный вес $w_i$-го компонента, который вычислялся по формулам:

$$w_i = \frac{\sum_{j=1}^{n} a_{ji}}{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}}, \sum_{i=1}^{n} w_i = 1$$

В случае больших размеров матрицы в целях получаются более точные оценки компонентов иерархии, вычисляется собственный вектор и собственные значения матрицы. В них используются следующие данные: $\lambda_{max}$– максимальное собственное значение матрицы $A$ $n$–порядка, $w_i$ – коэффициент относительного веса элементов матрицы $A$, $W = (w_1, w_2, \dots, w_n)$ – собственный вектор, которому соответствует $\lambda_{max}$. Общность решения задачи сравнения устанавливается соотношением $\alpha = \sum_{i=1}^{n} w_i$ и значением $\sum_{i=1}^{n} w_i = 1$. Если матрица $A$ имеет $n-1$ собственных значений $\lambda$, равных нулю и $\lambda_{max} = n$, то она является согласованной. При этом индекс согласованности $CI$ и коэффициент согласованности $CR$ вычисляется по формулам:

$$CI = \frac{\lambda_{max} - n}{n-1}, CR = \frac{CI}{E(CI)},$$

где $E(CI)$ – математическое ожидание матрицы парных сравнений $A(n \times n)$.

Критерий приемлемости парного сравнения элементов в матрицах размером $n \geq 3$ получен такой: $CR \leq 0{,}05$ и $CR < 0{,}1$ для $n > 5$. По результатам сравнения формируется квадратная матрица $F(k \times k)$. Аналогично проводится сравнение компонентов ПТС. В результате сравнения получается $k$ матриц. Возможный порядок каждой матрицы – $i$, а максимальный каждой из них – $m$.

Инструментом для сравнения является ExpertChoice входной матрицы $A$, которая автоматически вычисляет собственный вектор $W$, собственное значение $\lambda_{max}$ и коэффициент согласованности $CR$. Для вычисления $\lambda_{max}$ и $W$ используются соответствующие функции пакета MATLAB 6.5.

Результаты сравнений заносятся в форму, содержащую перечень весовых коэффициентов программ, критерии, индексы и коэффициенты согласованности. Они предоставляются в виде готовых результатов обработки матриц. Полученные весовые коэффициенты синтезируются с помощью пакета MATLAB 6.5. Результаты отображаются в виде отчета о распределении надежности по объектам системы.

**Прогнозирование плотности дефектов** на процессе ЖЦ проводится по модели RLM (Rome Laboratory Model) с помощью следующих действий.

1) Анализ значений параметров модели прогнозирования, включая остаток дефектов от предыдущего этапа работ с ПО системы, и используется для целевого распределения значения надежности.
2) Сравнение прогнозируемого значения надежности с распределенным значением.
3) Корректировки переменных параметров и учет текущего состояния системы.
4) Оценка параметров модели прогнозирования надежности.
5) Прогнозирование плотности дефектов.
6) Определение значений (допусков) для оценок результатов прогнозирования и анализа альтернатив.
7) Расчет прогнозного значения надежности системы.

Расчет плотности дефектов делается с помощью модели RLM и путем прогнозирования плотности дефектов по формуле:

$$D_0 = \prod_{i=1}^{9} K_i,$$

где $K_i$ – модификатор плотности дефектов $D_0$, с учетом пороговых значений данных о плотности дефектов.

Для каждой анализируемой системы результаты сравниваются с полученными по модели RLM. При проверке оказалось, что для ПО объемом 10-25 KSLOC погрешность прогнозирования плотности дефектов примерно составила 30-35%. Это объясняется некоторыми ограничениями системы Hugin Lite 6.5. Эти результаты используются при прогнозировании надежности компонентов.

**Прогнозирование надежности компонентов** реализуется по следующей формуле модели надежности:

$$R_i = exp\left[-D_i I_i \cdot \left(1 - exp\left(\frac{\rho_i K}{I_i \varphi_i} \cdot t\right)\right)\right],$$

где $\rho_i$ – параметр среды эксплуатации $i$-го компонента, $\varphi_i$ – характеристика среды разработки, $I_i$ – оцененный размер начального кода, а $D_i$ – прогнозируемая плотность дефектов в системе. Коэффициент дефектов $K$ – константа, полученная для всех объектов ПТС, а значения $\rho_i$ и $\varphi_i$ – взяты при первоначальном прогнозировании надежности, которые не изменяются во время разработки компонентов системы.

**Измерение надежности системы** выполняется согласно классификации дефектов (Orthogonal Defect Classification), в соответствии с которой каждый выявленный дефект использует параметры: тип дефекта, триггер дефекта, влияние дефекта. Эти параметры используются одной или двумя подходящими моделями надежности, из выше приведенных, в целях проведения оценки прогнозного значения надежности отдельных компонентов и системы в целом. В модели оценивания надежности (стандарт ICO/IEC 9126) заданы *атрибуты*, которые определяют способность системы преобразовывать исходные данные в результаты при условиях, зависящих от периода жизни системы (износ и старение не учитываются). Снижение надежности компонентов может происходить из-за ошибок проектирования.

К атрибутам надежности относятся следующие:

*Безотказность* – свойство системы функционировать без отказов (программ или оборудования). Если компонент содержит дефект, то во множестве $D = \{De | e \in L\}$ всех дефектов, можно выделить подмножество $E \subseteq D$, для которых результаты не соответствуют функции $F^m$, заданной в требованиях на разработку. Вероятность $p$ безотказного выполнения компонента на $De$, случайно выбранном из $D$ среди равновероятных, равна $1 - card\{E\}/card\{D\}$.

*Отказ* (failure) показывает отклонение поведения системы от предписанного выполнения предписанных ей функций. Появление отказа может быть причиной ошибки (fault/error), вызывающей его. Если ошибка сделана человеком, то используется термин mistake. Когда различие между fault и failure не критично используется термин defect, означающий либо fault (причина), либо failure (действие). Так как существует большое разнообразие видов отказов (внезапные, постепенные, сбои и др.), то определяется наработка на отказ, среднее время между появлением угроз и делается оценка ущерба, которая наносится соответствующими угрозами. Вычисление среднего времени $T$ наработки на отказ согласно стандарта реализуется по формуле:

$$T = \sum_{i=1}^{De} \nabla\, t_i^E \Big/ N,$$

где $\nabla t_i^E$ – интервал времени безотказной работы компонента $i$ –го отказа; $N$ – количество отказов в системе.

*Устойчивость к ошибкам и отказам*, которая показывает на способность системы выполнять функции при аномальных условиях (сбоях аппаратуры, ошибках в данных и интерфейсах, нарушениях в действиях исполнителей и др.) можно вычислить по формуле: $Y = N^v/N$, где $N^v$ – количество разных типов отказов, для которых предусмотрены средства восстановления; $N$ – общее количество всех отказов в системе.

*Восстанавливаемость* показывает способность возобновить функционирование системы после отказов для повторного исполнения и можно определить по формуле

$$T = \sum_{i=1}^{De} \nabla t_i^b \Big/ D,$$

где $\nabla t_i^b$ – время восстановления работоспособности компонента после $i$ – отказа; $D$ – количество дефектов и отказов в системе.

Количественная оценка надежности складывается согласно стандарту OSI/IEC 9126 из четырех атрибутов надежности системы по формуле:

$$Qv(reali) = \sum_{j=1}^{4} a_j\, m_j w_j,$$

где $a_j$ – атрибуты надежности, $m_j$ – метрики (внешние, внутренние), $w_j$ – весовые коэффициенты.

Если оценка надежности получена очень малая, то требуется устранить обнаруженные ошибки и тогда повторно измеряется надежность до возрастания надежность системы до требуемого уровня. Чем интенсивнее проводится эксплуатация, тем интенсивнее выявляются ошибки, и тем самым обеспечивается рост надежности.

Важным фактором, влияющим на оценку надежности ПО, являются – угрозы, приводящие к снижению безопасности системы и ущербности всей системы. В связи с возникающими киберугрозами в сети Интернет, в практике работы с системами появляются более изощренные методы борьбы с такими угрозами для обеспечения функциональной безопасности действующих разного рода прикладных систем [17, 19].

## *5. Перспективные задачи обеспечения безопасности и надежности*

Согласно ФЗ «О безопасности» (Федеральный закон от 28.12.2010 N 390-ФЗ (ред. от 05.10.2015)), ГОСТ Р 22.0.02, ГОСТ Р 51898-2002, ГОСТ 1.1-2002 и др. требуется обеспечение: функциональных и программно-технических интерфейсов при проектировании разных видов систем; безопасности и защищенности общества и государства от внутренних и внешних угроз; работоспособности и конкурентоспособности компьютерных систем, требующих прогнозирования рисков в сравнении с допустимыми рисками и принятие сбалансированных мер, противодействующих киберугрозам.

Основными направлениями развития являются:

- умные сети и города / технологии;
- интеллектуальные и онтологические системы;
- передовое производство компьютерных систем;
- робототехника и автономные системы - облачные вычисления;
- открытые данные и большие данные (Big Data);
- электронное управление;
- обмен метаданными по интероперабельным активам многократного использования и др.

Одним из важных направлений решения поставленных задач является безопасность и надежность систем, создаваемых на основе современных парадигм программирования.

## *6. Заключение*

Отдельные аспекты теории надежности компонентов систем начали исследоваться в проекте РФФИ № 16-01-00352-18. В нем реализованы методы создания отдельных компонентов систем с учетом особенностей моделей вариабельности; проведена экспериментальная конфигурационная сборка варианта ОС Linux и отработан подход к созданию систем и сайтов из готовых ресурсов и сервисов Интернета [1, 2, 20-27]. Готовые ресурсы описывались на языках программирования (C, C++, Basic, Java, Python и др.), а их интерфейсы на языке WSDL. Они трансформировались к выходному коду и могли обрабатывать передаваемые данные (простые, структурные и неструктурированные типы данных) в распределенной среде Интернет. Использовался формальный аппарат отображения (mapping) неструктурных типов GDT стандарта ISO/IEC 11404–2007 к фундаментальным типам данных [24] для сред VS.Net, IBMWebSphere, Linux и др. Обработка неструктурированных данных Web-приложений проводилась разными методами, в том числе Web Content Mining и ЯП среды W3C. К компонентам применялись средства конфигурации, верификации, тестирования и сбора данных для проведения оценки надежности программ, работающих с большими данными.

Исследования теоретических и практических основ обеспечения качества и безопасности технических и программных систем выполняются в новом проекте РФФИ №19-01-00206/19 «Модели, методы и средства надежности технических и программных систем», как дальнейшее развитие проекта 00352. В данной статье проведено исследование моделей надежности, качества и обеспечения безопасности компьютерных систем с учетом известных в России и за рубежом работ [4-7, 10-19]. Согласно стандарту ЖЦ разработка системы проводится путем проектирования компонентов, а их также верификации и тестирования со сбором данных о наличии ошибок и отказов. Дана характеристика процессов ЖЦ, на которых выполняется для компонентов сбор данных об ошибках, отказах и дефектах. Приведена таблица эксперимента по измерению надежности с использованием оценочных моделей (Мусы, Окомота, Шнайдервинда и др.) для малых, средних и больших программ с учетом данных собранных на процессах ЖЦ по распределению и прогнозированию надежности компонентов. Приведена стандартная модель оценки надежности и дана характеристика модели измерения надежности систем с учетом прогнозирования надежности и плотности дефектов с атрибутами и метриками стандарта изменения надежности систем. Определены перспективы развития безопасных и надежных систем.

## Список литературы / References

[1]. Лаврищева Е.М., Пакулин Н.В. Модели и методы надежности технических и программных систем. Материалы Пятой научно-практической конференции OS DAY, 2018 / Lavrishcheva E.M., Pakulin N.V. Models and methods of reliability of technical and software systems. In Proc. of the Fifth Scientific and Practical Conference OS DAY, 2018. Available at: http://0x1.tv/20180517F (in Russian), accessed 26.10.2019.

[2]. Лаврищева Е.М., Пакулин Н.В., Рыжов А.Г., Зеленов С.В. Анализ методов оценки надежности оборудования и систем. Практика применения методов. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 99-120 / Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue. 3, 2018, pp. 99-120 (in Russian). DOI: 10.15514/ISPRAS-2018-30(3)-8

[3]. Андон Ф.И., Коваль Г.И. и др. Основы инженерии качества программных систем. К.: Наукова думка, 2007, 670 стр. / Andon F. I. et al. Foundation of quality engineering software system. K.: Naukova Dumka, 2007, 670 p. (in Russian).

[4]. Липаев В.В. Надежность и функциональная безопасность комплексов программ реального времени. Москва, ЗАО «Светлица», 2013, 193 стр. / Lipaev V.V. Reliability and functional safety of software systems real time. Moscow, Svetlitsa, 2013, 193 p. (in Russian).

[5]. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии программных продуктов. К.: Наукова думка, 2009, 372 стр. / Lavrisheva E. M., Grishchenko V. N. Assembly programming. Foundation of software industries. K.: Naukova Dumka, 2009, 372 p. (in Russian).

[6]. Р.Т. Фатрелл, Д.Ф. Шафер, Л.И. Шафер. Управление программными проектами. Достижение оптимального качества при минимуме затрат. Москва, Санкт-Петербург, Киев, Вильямс, 2003, 1125 стр. / Robert T. Futrell, Donald F. Shafer, Linda Isabell Shafer. Quality Software Project Management. Prentice Hall, 2002, 1680 p.

[7]. Липаев В.В. Методы обеспечения качества крупномасштабных программных систем. М.: СИНТЕГ, 2003, 510 стр. / Lipaev V. V. Methods of quality assurance of large-scale software systems. M.: SINTEG, 2003, 510 p. (in Russian).

[8]. Буренин П.В., Девянин П.Н. Лебеденко Е.В., Проскурин В.Г., Цибцуля А.Н. Безопасность операционной системы общего назначения Astra Linux Special Edition. Москва, Горячая линия, 2018, 311 стр. / Burenin P.V., Devyanin P.N., Lebedenko E.V., Proskurin V.G., Tsibtsulya A.N. Security of the Astra Linux Special Edition general purpose operating system. Moscow, Goryatchaya Linaya, 2018, 311 p. (in Russian).

[9]. Зеленов С.В., Зеленова С.А. Моделирование программно-аппаратных систем и анализ их безопасности. Труды ИСПРАН, том 29, вып. 5, 2017 г., стр. 257-282 / Zelenova S.A., Zelenov S.V. Modeling and Risk Analysis of Hardware-Software Systems. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 5, 2017. pp. 257-282 (in Russian). DOI: 10.15514/ISPRAS-2017-29(5)-13.

[10]. Musa J.D. Okumoto K.A. Logarithmic Poisson Time Model for Software Reliability Measurement. In Proc. of the 7th International Conference on Software Engineering, 1984, стр. 230–238.

[11]. Shanthikumar J.G. Software realibility models: A Review. Microelectronics Reliability, vol. 23, issue 5, 1983, pp. 903-943

[12]. Yamada S., Ohba M., Osaki S. S–shaped software reliability grows modeling for software error detection. IEEE Transactions on Reliability, vol. R–32, № 5, 1983, стр. 475–478.

[13]. Chulani S. Constructive quality modeling for defect density prediction: COQUALMO. In Proc. of the International Symposium on Software Reliability Engineering, 1999, pp. 3-5.

[14]. Duval P., Matyas R., Grover A. Continuous integration improving Software quality and reducing risk. Addison Wesley, 2009, 691 p.

[15]. Горбенко А.В., Засуха С.А., Рубан В.И., Тарасюк О.М., Харченко В.С. Безопасность ракетно-космической техники и надежность компьютерных систем: 2000-е годы. Авиационно-космическая техника и технология, №1 (78), 2011, стр. 9-20 / Gorbenko A.V., Zasukha S.A., Ruban V.I., Tarasyuk O.M., Kharchenko V.S. The safety of rocket and space technology and the reliability of computer systems: the 2000s. Aerospace Engineering and Technology, №1 (78), 2011, pp. 9-20 (in Russian).Avizienis, J.-C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, 2004, pp. 11- 33.

[16]. Костогрызов А.И., Липаев В.В. Сертификация качества функционирования автоматизированных информационных систем. Москва, Изд-во "Вооружение. Политика. Конверсия", 1996, 275 стр. / Kostogryzov A.I., Lipaev V.V. Certification of the functioning quality of automated information systems. Moscow, Publishing House "Arms. Politics. Conversion", 1996, 275 p.

[17]. 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. Программирование, том 41, №1, 2015, стр. 44-67 / I.S. Zakharov, M.U. Mandrykin, M.U. Mandrykin, V.S. Mutilin, E.M. Novikov, A.K. Petrenko, A.V. Khoroshilov. Programming and Computer Software, vol. 41, №1, 2015, pp. 49–64

[18]. Абросимов Н.В., Костогрызов А.И. и др. Безопасность России. Правовые, социально-экономические и научно-технические аспекты. Техногенная, технологическая и техносферная безопасность. М.: МГОФ «Знание», 2018, 1016 стр. / Abrosimov N.V., Kostogryzov A.I. et al. Security of Russia. Legal, socio-economic and scientific-technical aspects. Technogenic, technological and technosphere safety. M, Znanie, 2018, 1016 p. (in Russian).

[19]. Кулямин В.В., Лаврищева Е.М., Мутилин В.С., Петренко А.К. Верификация и анализ вариабельных операционных систем. Труды ИСП РАН, том 28, вып. 3, 2017, стр. 189-208 / 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-2016-1(2)-12.

[20]. Lavrischeva E.M., Mutilin V.S., Ryzhov A.G. Designing variability models for software, operating systems and their families. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 5, 2017, pp. 93-110. DOI: 10.15514/ISPRAS-2017-29(5)-6.

[21]. 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.

[22]. Козин С.В. Конфигурационная сборка варианта ядра Linux для прикладных систем. Труды ИСПРАН, том 30, вып. 6, 2018 г., стр. 161-170 / Kozin S.V. Linux kernel configuration build for application systems. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 6, 2018, pp. 161-170 (in Russian). DOI: 10.15514/ISPRAS-2018-30(6)-9.

[23]. Лаврищева Е.М., Рыжов А.Г. Подход к созданию систем и сайтов из готовых ресурсов. Труды XX Всероссийской научной конференции «Научный сервис в сети Интернет», 2018, стр. 321-346 / Lavrishcheva E.M., Ryzhov A.G. Approach to creating systems and websites from ready-made resources.: In Proc. of the XX All-Russian Scientific Conference on Scientific service on the Internet, 2018, p. 321-346 (in Russian).

[24]. Лаврищева Е.М. Компонентная теория и коллекция технологий для разработки индустриальных приложений из готовых ресурсов. Труды 4-й научно–практической конференции «Актуальные проблемы системной и программной инженерии», 2015 г., стр. 101–119 / Lavrischeva E.M. Component theory and collection technology for development of industry application from ready resources. In Proc. of the 4th scientific and practical conference on Actual Problems of System and Software Engineering, 2015, pp. 101-119 (in Russian).

[25]. E.M. Lavrischeva. The Scientific basis of software engineering. International Journal of Applied and Natural Sciences, vol. 7, 2018, pp. 15-32.

[26]. Lavrischeva E.M. Science of the computer programs and systems in XX-XXI centuries: past, present, future. European Journal of Mathematics and Computer Science, vol. 5, no. 1, 2018, стр. 67-92.

## Информация об авторах / Information about authors

Екатерина Михайловна ЛАВРИЩЕВА, доктор физико-математических наук, профессор, главный научный сотрудник ИСП РАН, профессор МФТИ. Научные интересы: технология программирования, программная инженерия.

Ekaterina Mikhailovna LAVRISCHEVA, Doctor of Physics and Mathematics, Professor, Chief Researcher at ISP RAS, Professor at MIPT. Research interests: programming technology, software engineering.

Сергей Вадимович ЗЕЛЕНОВ, кандидат физико-математических наук, старший научный сотрудник ИСП РАН, доцент ВШЭ. Научные интересы: модели программно-аппаратных систем, надежность программного обеспечения.

Sergey Vadimovich ZELENOV, Candidate of Physics and Mathematics, Senior Researcher at ISP RAS, Associate Professor at HSE. Research interests: models of software and hardware systems, software reliability.

Николай Витальевич ПАКУЛИН, кандидат физико-математических наук, технический директор компании Pax Datatech PTE, Ltd. Научные интересы: сетевые технологии, блокчейн.

Nikolai Vitalievich PAKULIN, Candidate of Physics and Mathematics, Technical Director of Pax Datatech PTE, Ltd. Research: network technologies, blockchain.

**DOI: 10.15514/ISPRAS-2019-31(5)-8**



# DOOR: Подход к реструктуризации распределенных объектно-ориентированных систем на основе нейронных сетей

*А. Хан, ORCID: 0000-0003-4355-484X <soft_engr_isb@yahoo.com>*

*Университет COMSATS Исламабад,*
*Пакистан, 45550, Исламабад, Парк Роад, Тарлай Калан*

**Аннотация.** Для развития распространенных программных систем инженерами и дизайнерами в настоящее время используется объектно-ориентированный подход, который помогает строить распределенные объектно-ориентированные системы. Отличительной особенностью распределенных объектно-ориентированных систем является компетентное распределение классов объектов по различным узлам. Обычно информация о распределении классов по серверам в приложениях отсутствует, что стимулирует проведение процедуры реструктуризации, позволяющей повысить производительность. В предлагаемом подходе реструктуризация программного обеспечения систем осуществляется с помощью адаптивного метода с использованием нейронной сети. На начальном этапе создается граф зависимости классов, узлы в котором представляют классы, а связи между узлами соответствуют зависимостям между классами. Затем извлекаются свойства классов, входящих в этот граф, которые передаются в качестве входных данных в нейронную сеть для ее обучения. После этого на основе учета зависимостей классов выполняется кластеризация, приводящая к разбиению множества классов распределенной объектно-ориентированной системы на слабосвязанные подмножества. Далее создается граф кластеров, вершины в котором соответствуют кластерам, а ребра – каналам связи, которые могут существовать между кластерами. К полученному графу применяется алгоритм k-medoids, который используется для сбора кластеров таким образом, что количество собранных групп кластеров становится равным количеству доступных узлов системы. Сформированные в результате группы кластеров оказываются слабосвязанными. В завершение группы кластеров приписываются различным доступным узлам в распределенной среде. Результаты моделирования показали, что предлагаемая работа дает более эффективные результаты по сравнению с существующими методами.

**Ключевые слова:** распределенные объектно-ориентированные системы; диаграмма зависимости классов; рекурсивная кластеризация графов; слабое связывание; нейронная сеть; распределенная архитектура

**Для цитирования:** Хан А. DOOR: Подход к реструктуризации распределенных объектно-ориентированных систем на основе нейронных сетей. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 109–126. DOI: 10.15514/ISPRAS–2019–31(5)–8

# DOOR: Distributed Object Oriented Software Restructuring Approach Using Neural Network

*A. Khan, ORCID: 0000-0003-4355-484X <soft_engr_isb@yahoo.com>*
*COMSATS University Islamabad,*
*Park Road, Tarlai Kalan, Islamabad, 45550, Pakistan*

**Abstract:** To develop common software systems, engineers and designers are currently using an object-oriented approach that helps build distributed object-oriented systems. A distinctive feature of distributed object-oriented systems is the distribution of classes of objects among various nodes. Typically, there is no information about the distribution of classes across servers in applications, which encourages a restructuring procedure that improves productivity. In the proposed approach, software restructuring of distributed object-oriented systems is carried out using the adaptive method using a neural network. At the initial stage, a graph of class dependencies is created, in which nodes represent classes, and relations between nodes correspond to dependencies between classes. Then, the properties of the classes included in this graph are extracted, which are transmitted as input to the neural network for its training. After that, based on the account of class dependencies, clustering is performed, leading to the partition of the set of classes of the distributed object-oriented system into loosely coupled subsets. Next, a graph of clusters is created, the vertices in which correspond to clusters, and the edges correspond to communication channels that may exist between clusters. The k-medoids algorithm is applied to the resulting graph, which is used to collect the clusters in such a way that the number of collected cluster groups becomes equal to the number of available system nodes. The resulting cluster groups are loosely coupled. Finally, cluster groups are assigned to various available nodes in a distributed environment. The simulation results showed that the proposed work gives more effective results compared to existing methods.

**Keywords:** distributed object oriented systems; class dependency graph; recursive graph clustering; low coupling; neural network; distributed architecture

**For citation:** Khan A. DOOR: Distributed Object Oriented Software Restructuring Approach Using Neural Network. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019 г., pp. 109-126 (in Russian). DOI: 10.15514/ISPRAS–2019–31(5)–8

## *1. Введение*

С развитием технологии использование новейших программных приложений существенно способствовало и научному прогрессу, и экономическому росту. К сожалению, одновременно растут потери из-за намеренных или ненамеренных погрешностей в программном обеспечении, что ставит перед разработчиками программного обеспечения крайне актуальную задачу повышения качества создаваемых ими программных продуктов [1]. Проверки качества программных продуктов осуществляются на основе анализа исходных кодов программного обеспечения. Повысить качество программ удается совместным анализом качества структуры программ и ее функциональных возможностей. Обеспечение качества программ представляет собой заметную часть процесса разработки программного продукта [2].

Постоянный рост сложности, критической важности и безальтернативности использования программных систем способствует повышенному вниманию к качеству программного обеспечения при его разработке. Чтобы удостовериться в том, что система не имеет ошибок, она должна быть тщательно протестирована, хотя во многих случаях прямое тестирование затруднительно. Это приводит к созданию специальных групп разработчиков, занятых сопровождением программных систем, которые целиком сосредоточены на тестировании программных систем [3]. Рост скорости разработки программных продуктов приводит к серьезным конфигурационным изменениям,

программные функции меняются под влиянием сиюминутных обстоятельств, что приводит к столь же серьезным изменениям самой сути программных продуктов [4].

Конфигурация программного обеспечения наряду с его качеством часто становится жертвой ограниченного бюджета. Использование таких прекрасных языков программирования, как C#, Java и других не может отменить необходимости строить четкое и понятное описание структуры программы [8]. Как правило, методы реструктуризации приводят к улучшению исходного кода, повышая его качество, выявляя неописанные функции, иногда даже до первоначального выпуска продукта, делая его элементы более понятными и более пригодными для повторного использования [5].

Рефакторинг связан с перестройкой программного обеспечения. Этим термином обозначается изменение в программном обеспечении, выполняемое для улучшения его внутреннего качеств без изменения внешнего поведения. Основной целью этого процесса является повышение нефункциональных особенностей кода, некоторых его выбранных качеств, его понятности и гибкости [6].

Рефакторинг и поддержка программ – это два разных процесса. Рефакторинг в основном связан с повышением разумности уже существующего и действующего программного обеспечения. Рефакторинг отличается от сопровождения тем, что сопровождение призвано менять поведение программного обеспечения, то есть выявлять и устранять ошибки, ускорять работу приложения или как-то улучшать функциональность [7]. В то же время, рефакторинг программного обеспечения улучшает внутреннее качество программного обеспечения, не влияя на его внешнее поведение. Следовательно, он является эффективным и сравнительно безопасным способом улучшения качества программного обеспечения [9].

Совершенствование программной среды представляет собой длительный и непрерывный процесс. Программная среда, как правило, в течение некоторого времени претерпевает последовательность как небольших, так и больших изменений [10]. Чтобы найти, а потом и усовершенствовать методику разработки программного обеспечения и разрабатываемые программные продукты, были созданы и утверждены стандарты разработки программного обеспечения. Среди предложенных моделей в течение уже многих лет заметно выделяется объектно-ориентированный подход [11]. Эта парадигма рассматривает программу как совокупность взаимодействующих объектов, каждый из которых скорее заботится о хранении своего индивидуального частного состояния, чем о предоставлении средств хранения глобального состояния. Основу парадигмы объектно-ориентированного программирования составляют понятия динамического связывания, инкапсуляции, наследования и полиморфизма. Объекты стало возможность распределять по узлам системы, где потом они могли обрабатываться либо параллельно, либо последовательно, в зависимости от конкретики объектно-ориентированного приложения [12].

Революционная парадигма программного обеспечения, называемая объектно-ориентированным программированием (ООП), представляет собой метамодель, построенную на использовании понятия о классах и об экземплярах этих классов на основе теории абстракции, и механизмов инкапсуляции, наследования, полиморфизма и других. Эти теоретические построения способствуют созданию повторно используемых и способных к развитию программных модулей [13]. Объектно-ориентированный подход, первоначально возникший как парадигма программирования, в настоящее время используется специалистами и проектировщиками для решения сложных задач во многих научных областях.

Основным этапом моделирования распределенной объектно-ориентированной методологии (РОО) является выявление области видимости объекта. Этот этап уменьшает потребность в установлении связей с объектом в тех местах, где он недоступен [14].

Приложение РОО создается с использованием различных программных платформ, которые обеспечивают такие свойства, как мобильность, соответствие требованиям и способность к взаимодействию. На этих стадиях проявляются базовые свойства программных платформ и их конфигураций, которые влияют на архитектуру и полезность создаваемых систем [15]. Первые возникающие варианты приложения РОО, безусловно, не обладают достаточным качеством. Чтобы его добиться, можно действовать двояким образом: для взаимодействия с программными компонентами можно реконфигурировать оборудование, но чаще для взаимодействия с аппаратурой изменяется структура программ [14].

Настоящая работа построена следующим образом: в разд. 2 рассматриваются родственные работы, касающиеся предлагаемого нами метода, в разд. 3 кратко обсуждается предлагаемая методика, разд. 4 посвящен оценке результатов исследования, в разд. 5 обобщаются результаты, описанные в статье.

## *2. Обзор литературы*

В работе Айхуа Гу и соавторов [16] предлагается использовать метрику связности сложных сетей (Cohesion Based on Complex Networks, CBCN), разработанную главным образом на основе расчета коэффициента средней кластеризации класса (Class Average Clustering, CAC) с помощью диаграмм, демонстрирующих причины объединения в кластере нескольких классов. Кроме того, метрика CBCN была оценена на гипотетическом примере сравнения с метрикой объединения классов (Class Union Measurement, CCM) по четырем свойствам (связывающие модули, неотрицательность и нормализация, монотонность, пустые и максимальные значения). Показатели метрики CBCN, основанные на корреляции информации с семнадцатью наилучшими из типичных CCM систем объединения классов, оказались наилучшими для всех остальных. Применение метрики CBCN к трем программам с открытым исходным кодом вычисления коэффициентов CAC, показало, что класс, требовавший модификации, усложнения и поддержки в системе с открытыми кодами, был протестирован хуже остальных. У упомянутых систем имелось степенное распределение коэффициентов CAC, что обеспечило дальнейшее понимание CBCN.

В работе Немитари Ажиенка с соавторами [17] была сделана попытка связать эффективность измерения семантического связывания (Semantic Coupling, SC) классов объектно-ориентированных программ, используя 1) простые методы, ориентированные на идентификаторы, и 2) полные корпуса классов в программной системе. Впоследствии они исследовали взаимодействие между семантическим и изменяемым связыванием (Change Coupling, CC). Что касается отношений между логическими и семантическими зависимостями, то в 79 объектно-ориентированных проектах и проектах с открытым кодом не было выявлено никакой связи между логическими и семантическими условиями и качеством программных продуктов. Тем не менее, они признали взаимовлияние семантических и логических условий. Было отмечено, что более семидесяти процентов семантически связанных классов регулярно развивались, а классы, связи которых, хотя бы в некоторой степени, менялись, обычно обладали семантическими связями. Результаты показали, что: (a) методы, ориентированные на использование идентификаторов классов, более эффективны, хотя и не во всех случаях (только при рассмотрении семантически очень похожих классов), (b) между семантическими и логическими зависимостями не было никакой линейной связи. Кроме того, они обнаружили, что (c) между ними существуют взаимоотношения, так как более семидесяти процентов семантически связанных классов обычно развиваются совместно, а классы, связанные через CC, обычно, хотя бы немного, связаны и семантически.

В работе Нимиша Кумара и соавторов [18] представлена общая концепция метода управления качеством многочисленных свойств объектно-ориентированной структуры, который логически рассматривается как процедуры качества с вложенными в них процедурами, зависящие от перспектив развития сетевых приложений. Ключевым посылом выработки этих рекомендаций было изучение и распространение ранее разработанных процедур управления качеством на сетевые приложения. Этот метод был выбран из-за не всегда продуманного и очень разного характера сетевых приложений. Для управления структурой программной системы использовались рекомендации стандарта ИСО/МЭК 9126. Универсальность метода доказывается тем, что программирование обычных и сетевых приложений предлагается вести сходными средствами. При этом, конечно же, в зависимости от вида приложения (сетевое или обычное) можно добавлять, изменять или удалять различные его свойства. Еще больше повышает точность измерения показателя качества исследуемой структуры увеличение глубины иерархии и количества узлов в ней (сочетанием параметров качества с процедурами, вложенными процедурами, показателями и их отдельными частями). Таким образом, полученная модель оказалась достаточно гибкой и пригодной для использования со всеми программными приложениями и фундаментальными технологиями.

Аншу Парашар и Джитендер Кумар Чхабра [19] представили оригинальные оценки, предназначенные специально для классов: набор изменений-воздействий классов и индекс изменяемого связывания (CC). Сначала эти оценки используются для вычисления индекса CC классов, затем, рассчитываются показатели для классов. Предложенная структура прогнозирования изменчивости давала разработчикам большое количество различной информации, в частности, прогнозируя уровень изменчивости связывания класса. Полезность подхода была подтверждена путем выявления структурных особенностей классов и вычисления индекса CC, а также проведением достаточно точных замеров влияния измененных воздействий на изменения в классах. Предложенные меры были подтверждены эмпирически, были даны ответы на конкретные запросы исследователей, что подтвердило разумность потока изменений. Полученные результаты оказались достаточно надежными, они указывали на то, что предлагаемый прогноз потока потенциальных изменений крайне полезен при поддержке программных систем. Была подтверждена достоверность введенных мер изменчивости, что явилось реакцией на другие запросы, возникающие в ходе конкретных исследований. В частности, потоки информации об изменениях, зафиксированные как история изменений классов, могут быть задействованы для обнаружения изменяемого связывания, что помогает визуализировать предполагаемые архитектурные изменения.

Дарио Ди Нуччи с соавторами [20] исследовали программные компоненты, подвергавшиеся изменениям как со стороны специально выделенных для сопровождения разработчиков, так и со стороны других разработчиков, вносивших изменения в программы. В этой статье описано и другое исследование, в котором изучались ожидания разработчиков в отношении возможных видов ошибок, в частности, они выделяли 2 подхода i) упор на количество фрагментов кода, как основу для поиска ошибок (разработчик вносит изменения в те фрагменты, которые используются в большем числе распределенных узлов) ii) упор на количество соглашений, которые должны соблюдаться в тех или иных узлах (семантическое рассеяние). Указанные подходы, которые были названы предсказателями ошибок, были использованы в эксперименте с привлечением двадцати шести программных систем с открытым исходным кодом. Достигнутые результаты подтвердили действенность модели авторов, они также показали направления расширения и дополнения использованной стратегии. Была получена и затем протестирована на одиннадцати прогнозах гибридная предсказательная модель и пять альтернативных методов. Полученные результаты показали, что (a) гибридная модель продемонстрировала более высокую точность по сравнению с каждой отдельно взятой из

пяти стратегий; (б) разработанные авторами предсказатели ошибок вносят решающий вклад в самые эффективные «гибридные» предсказательные модели.

Вакар Аслам и Фара Иджаз [21] рекомендовали методологию, названную ими распределением задач (task allocation). Работа по этой методологии выполняется в 2 этапа: на этапе 1 определяются условия и перспективы, влияющие на приписку, на этапе 2 предлагается поддающаяся измерениям система, раздающая задания ближайшим коллегам пользователя наилучшим образом. Реквизиты задач формулируются как возможности, рассматриваемые в самых разных аспектах, учитывающих, например, специальные индивидуальные условия. Одним из важных решений считается распределение задач между членами группы разработчиков. Это решение обычно принимается субъективно, а не детерминированными средствами, поэтому связанные с ним риски могут сказываться на результатах работ, выполняющихся впоследствии. Предлагаемый авторами метод учитывает потребности каждой отдельной работы из тех, которые надо будет выполнять при реализации проекта. Для всех и каждой отдельной работы из членов команды выбирается наилучший исполнитель, причем выбор основывается на знании длительности аналогичных работ, на знании методов оценки окончания сравнимых работ, а также на предположениях об относительной безошибочности результата. Авторский метод напоминает методику фокусирования на целях и наилучшего отождествления. Другие предложения авторов приводят к улучшению соотношения цена-качество, а также к достижению многих других целей. Все это позволяет проводить апостериорную оценку качества распределения задач по разработчикам и, в конечном итоге, снижать опасность от связанных рисков.

## *3. Реструктуризация распределенных объектно-ориентированных программ с использованием нейронных сетей*

Методы объектно-ориентированного программирования (ООП), предоставляя передовую платформу для разработки программного обеспечения, позволяют легко и просто создавать программные приложения. На основе подходов распределенных объектно-ориентированных (РОО) систем созданы многочисленные прикладные системы, позволяющие решать сложные задачи в различных научных областях. Центральным аспектом РОО систем является организация адекватного распределения классов, определенных в программе, по разным структурным узлам, то есть снижение риска возникновения несоответствия, которое может возникнуть, если структура программного обеспечения не соответствует структурной организации используемого оборудования.

Для повышения производительности РОО системы предлагается использовать адаптивный метод, использующий возможности нейронных сетей (НС), на основе которого может осуществляться реструктуризация программного обеспечения. Применение этого метода позволяет уменьшить общее число кластеров классов, которые изначально создаются при обучении нейронной сети, их уменьшение соответственно уменьшает потребность в выделенных вычислительных ресурсах.

Первым шагом предложенного метода является создание графа зависимостей классов (ГЗК), в котором узлы соответствуют классам, а связи между ними представляют зависимости между классами. После этого компоненты объектов, методов, переменных, файлов включения и объема кода, связанные с классами в ГЗК, извлекаются из их определений и передаются в качестве входных данных в нейронную сеть как исходные данные для ее обучения. Далее выполняется кластеризация, с помощью которой ОО-система сегментируется на слабосвязанные подсистемы с использованием метода кластеризации на основе зависимостей классов (Class Dependency Based Clustering, CDBP). Объединенные в кластеры классы включаются в кластерные графы с использованием техники k-medoids, и, наконец, используя рекурсивную кластеризацию k-

средних, созданные кластерные структуры сопоставляются с фиксированной конфигурацией доступных машин в целевой распределенной архитектуре. Структура предложенной методологии поясняется на рис. 1.

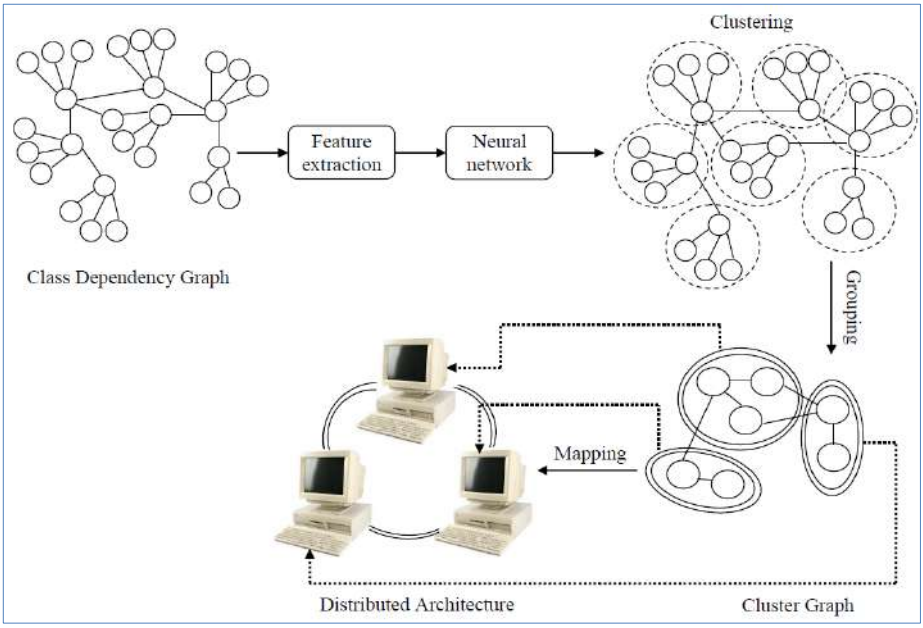


*Рис. 1. Предлагаемый подход*
*Fig. 1. The Proposed Methodology*

## 3.1 Граф зависимостей классов

Процесс первичной оценки связи между классами в РОО системе заключается в приписке каждого отдельного класса программы некоторому узлу распределенной системы. В совокупности связи между классами, размещенными в тех или иных узлах, составляют граф зависимостей классов объектно-ориентированной системы. Пример ГЗК показан на рис. 2.

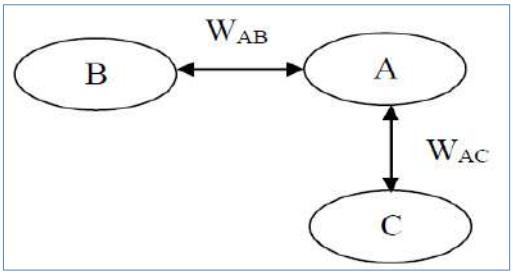


*Рис. 2. Граф зависимостей классов.*
*Fig. 2. Class Dependency Graph*

Каждая вершина графа ГЗК обозначает некоторый класс, а ребра на графе между классами A и B соответствуют наличию связи, образующей либо при передаче данных между объектами классов, либо при наличии некоторого отношения между классами. Вес ребра $W_{AB}$ вычисляется как стоимость коммуникационных событий между классом A и классом B. Если между классами нет выявленных зависимостей или отношений, и между ними не никогда не осуществляется передача данных, на ГЗК между классами никаких ребер не возникает.

## 3.2 Извлечение свойств

После создания ГЗК на основе анализа текста программы выявляются свойства объектов, переменных, методов, файлов включения и объема программы, относящихся к выделенным классам в ГЗК.

- Объект: Объект выступает как центральная единица ООП. Обычная Java-программа содержит множество определений взаимосвязанных объектов, взаимодействующих друг с другом посредством определенных в программе методов классов. Каждый объект характеризуется состоянием, поведением и идентичностью.
- Переменные: Объектно-ориентированная парадигма, основанная на классах и объектах, как экземплярах классов, допускает наличие в классах «статических» переменных, которые существуют в единственном для всех объектов класса экземпляре и никогда не являются принадлежностью какого-либо одного из объектов. Такая переменная выступает в качестве отдельной формы атрибута класса (поля данных или, иначе, свойства класса, его информационной части).
- Метод: Процедура, которая является принадлежностью объекта и может генерировать сообщения, называется в объектно-ориентированной парадигме методом. На самом деле это фрагмент программы, который содержит последовательность операторов, выполняющих задачу. Методы обладают интерфейсом, который используется другими классами для изменения и доступа к свойствам данных объекта.
- Файлы включения: Файлы импорта используются для передачи пользовательских и встроенных системных определений в исходный файл Java, чтобы отдельный класс мог обращаться к классу другого кластера, напрямую используя его имя.
- Число строк: общее количество строк кода, содержащихся в программе.

## 3.3 Нейронная сеть

Выявленные на предыдущем этапе элементы передаются нейронной сети в качестве входных данных процесса обучения. Обучаясь, нейронная сеть корректирует собственные веса, базируясь на наборах входных данных и выходные с предсказанными целевыми значениями.

Веса синапсов постепенно меняют свои значения, пока сеть не стабилизируется. Каждая итерация выполняется в двух направлениях: прямая передача сигнала для достижения решения, а также обратное распространение ошибок, выполняемое для преобразования весов. Такие прямые и обратные передачи сигналов выполняются до тех пор, пока решение состояние нейронной сети не будет признано приемлемым с учетом ранее установленного допуска. Структура нейронной сети показана на рис. 3. Действия по обратному распространению ошибки в процессе обучения сети показаны на следующих рисунках.

**Шаг 1:** Выбрать случайные веса в интервале [0, 1] и передать их на выход, а также на нейроны скрытого слоя. Передать веса на все нейроны информационного слоя.

**Шаг 2:** Классификатор сам выбирает набор обучающих данных в качестве входных, ошибка оценивается как

$$BP_{err} = Z_{tar} - Z_{out}. \quad (1)$$

В формуле (1) $Z_{tar}$ является эталонным значением выхода, реальным выходом сети является значение $Z_{out}$, которое определяется как $Z_{out} = [Z_2^{(1)}\ Z_2^{(2)}\ \dots\ Z_2^{(N)}]$. Выходы сети $Z_2^{(1)}, Z_2^{(2)}, \dots, Z_2^{(N)}$ представляются как

$$Z_2^{(l)} = \sum_{r=1}^{N} W_{2rl} Z_l(r), \quad (2)$$

где

$$Z1(r) = \frac{1}{1 + \exp(-W_{1lr}\, Z_{ln})}. \quad (3)$$

Формулы (2) и (3) вводят функции активации, реализованные в выходном и в скрытом слоях соответственно.

**Шаг 3:** Изменить все веса нейронов на величину $\Delta W$, где $\Delta W$ вычисляется как

$$\Delta W = \gamma \times X2 \times BP_{err} \quad (4)$$

В формуле (4) величина $\gamma$ обозначает скорость обучения, обычно она находится в диапазоне от 0.2 до 0.5.

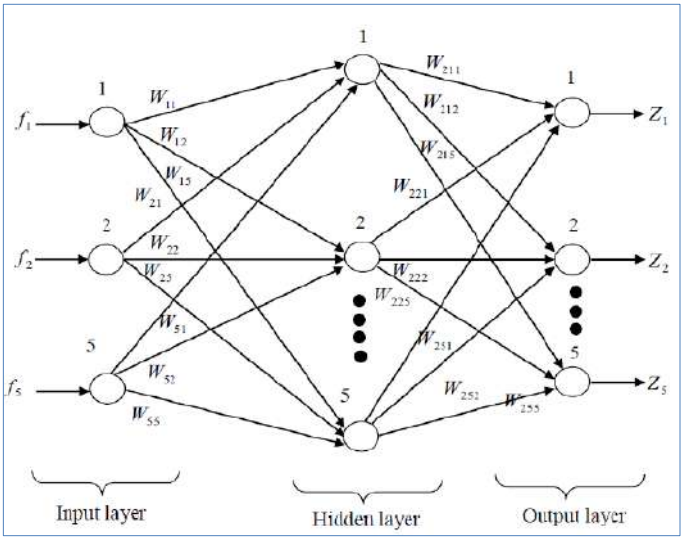


*Рис. 3. Структура нейронной сети*
*Fig. 3. Structure of Neuron Network*

**Шаг 4:** Повторять описанный процесс, начиная с шага 2, пока ошибка $BP_{err}$ не будет уменьшена до минимального значения. Обычно в качестве критерия используется соотношение $BP_{err} < 0.1$.

Уточненные в процессе обучения функции, использовавшиеся для кластеризации, уменьшают начальную совокупность кластеров нейронной сети и, таким образом, уменьшают совокупность выделенных ресурсов.

## 3.4 Кластеризация системы классов

После завершения этапа обучения сети с помощью алгоритма CDBP выполняется разделение программной системы на слабосвязанные подсистемы. На рис. 4 показан псевдокод кластеризации на основе зависимостей классов. С точки зрения языка программирования Java кластер представляет собой пространство имен, в котором собраны близкие друг к другу классы и их интерфейсы. До некоторой степени кластеры напоминают специальные папки, размещенные в архиве персонального компьютера. Наличие кластеров стимулирует разработчиков группировать классы (а также интерфейсы) вместе. Все эти классы будут каким-то образом связаны, так что все они могут понадобиться при решении определенного набора задач. Поскольку целью является объединение зависимых классов, кластеризация с использованием алгоритма CDBC, показанная на рис. 4, может оказаться весьма полезной. Реализация алгоритма CDBC разбивает программу на несколько кластеров, в которые входят взаимодействующие и

схожие классы. Сначала производится идентификация классов, а затем формируются кластера.

**Идентификация классов объектно-ориентрованных программ**: Перед выполнением процедуры кластеризации производится текстовый анализ программной системы и выделяются все использованные в программе классы. В нашем случае использовался программный продукт, созданный на языке Java, поэтому текстовый анализ выполнялся над файлами с расширением java.

**Формирование кластера**: После распознавания классов объектов алгоритм CDBC способен выявлять группы зависимых классов. Для очередного класса проверяется, содержатся ли в исходном коде операции создания объектов этого класса. Если да, класс помещается в кластер в соответствии со своими зависимостями. Если условие не выполняется, происходит переход к анализу следующего класса. Пополнение кластеров продолжается до тех пор, пока каждый класс, используемый в анализируемой программе, не будет отнесен к тому или иному кластеру.

```
Requirement: PackageContainer=0, PackageName=0,
ProgramClassList=0, NumOfObjects=0, NumOfImports=0

For each PackageContainer do
   For each File do
        If fileNameExtension= .Java then
           Add File to ProgramClassList
        End if
   End for
End for

For each ProgramClassList do
   Read Program File
   Search each class name in Header && SearchObject for ClassHeader
        If Class contains ClassHeader and its Object then
           Group class under the Dependency package
        Else
           Search in another ProgramClassList
        End if
End for

For each Package in PackageContainer do
   If NumOfClassInPackage<NumOfObjects then
        Add to Clusters
   End if
End for
```

*Рис. 4. Псевдокод кластеризации на основе зависимости классов*
*Fig. 4. Pseudocode for Class Dependency Based Clustering*

## 3.5 Группировка кластеров

Кластеры классов, созданные на предшествующем этапе, используются в дальнейшем в качестве кандидатов для распределения по узлам системы. Предлагаемый метод заключается в объединении кластеров в группы, что обеспечивает снижение затрат на организацию связей между ними. Чтобы добиться поставленной цели, создается граф кластеров, вершины в котором соответствуют кластерам, а ребра – каналам связи, которые могут существовать между кластерами. К полученному графу применяется алгоритм k-medoids, который используется для сбора кластеров таким образом, что количество собранных групп кластеров становится равным количеству доступных узлов системы. Сформированные в результате группы кластеров оказываются слабосвязанными. В

завершение группы кластеров приписываются различным доступным узлам в распределенной среде.

### 3.5.1 Алгоритм k-medoids

Группировка применяется для объединения кластеров классов с целью собрать в одной группе кластеры, классы которых некоторым образом более похожи друг на друга, чем на классы из других кластеров. Для группировки кластеров используется стратегия кластеризации k-medoids. Медоид – это элемент кластера, отличие которого от других элементов этого же кластера незначительно. Тем самым, медоиды, атрибуты которых сильно отличаются от средних значений, должны исключаться из наборов данных, в которых остаются только элементы, наиболее сильно приближенные к средним значениям. Псевдокод для алгоритма кластеризации k-medoids показан на рис. 5.

**Input:**
$K$, Number of cluster groups
**Output:**
$K$ Set of Cluster Groups

**Begin**
Arbitrarily choose $K$ objects as the initial representative objects
**Repeat**
- Assign each remaining object to the cluster with the nearest representative object.
- Randomly select a non-representative object $o_{rand}$
- Compute total cost $S$ of swapping representative object $o_s$ with $o_{rand}$

**If** $S < 0$ **then**
- Swap $o_s$ with $o_{rand}$ to form the new set of $K$ representative objects

**Until** No change
**End**

*Рис. 5. Псевдокод алгоритма k-medoids*
*Fig. 5. Pseudocode for k-medoids Algorithm*

## 3.6 Распределение классов по узлам сети

Построенные группы кластеров, используемых в прикладной РОО-системе классов теперь надо сопоставить с узлами серверной сети, что должно способствовать достижению более высокой производительности. Процедура сопоставления пишется таким образом, чтобы максимально ограничить зависимость классов от классов других групп и снизить любой обмен информацией между ними. Обычно распределенная система, для которой выполняется такая работа, однородна, то есть состоит из одинаковых процессоров, взаимодействующих в сети на основе единых принципов обмена данными. В предлагаемой стратегии сопоставление классов и узлов серверной сети выполняется на основе алгоритма рекурсивной кластеризации k-средних.

### 3.6.1 Рекурсивный алгоритм кластеризации k-средних

Предложена новая итеративная стратегия, использующая алгоритм k-средних и рекурсивно строящая последовательности кластеров. Цель этих действий заключается в том, чтобы получить решение k-средних для полного набора данных путем рекурсивной актуализации взвешенной формы k-средних над растущим, но все же небольшим числом

представителей кластера. Псевдокод для алгоритма рекурсивной кластеризации k-средних показан на рис. 6.

На начальном этапе итерационного процесса набор данных разбивается на различные непересекающиеся подмножества, называемые блоками, которые собраны в гиперкубах одинакового размера. Для каждого блока затем создается представитель и вычисляется характеристика, называемая весом. Наконец, над группой представителей выполняется адаптированный вариант алгоритма Ллойда с весами. Шаг за шагом, начиная с полученного разбиения, создаются уточнения, в которых каждый гиперкуб заменяется на другой гиперкуб с тем же весом.

Выполнение алгоритма позволяет заменять подмножества кластеров предыдущей итерации другими кластерами так, чтобы общая ошибка уменьшалась. Этот процесс повторяется до тех пор, пока не будет выполнено заранее заданное число шагов, либо пока очередная итерация не будет отличаться от предыдущей слишком незначительно. Выполняемые вычисления полностью следуют варианту алгоритма Ллойда с весами элементов, называемого взвешенным алгоритмом Ллойда, при работе которого для каждой группы представителей определенного сегмента исследуется вес, рассчитываемый для каждой группы в ее совокупности. По завершении работы алгоритма построенные в результате группы кластеров отображаются на доступные серверные узлы.

**Input:** Number of Clusters $K$, Integers $\{q_{\min}, q_{\max}\}$, threshold $\eta$
**Output:** Initial set of centroid

**Begin**
**For** $q = q_{\min} : q_{\max}$ **do**
- Construct the partition $P_q$
- Define the weight set $W_q$
- Update the centroid set approximation $C_q = \{c_j^q\}_{j=1}^K : C_q = Weighted\_Lloyd(W_q, K, C_{q-1})$
- Compute the centroid set displacement measure $d(C_{q-1}, C_q) = \max_{j=1,2,\dots,K} \left\| c_j^q - c_j^{q-1} \right\|^2$

**If** $d(C_{q-1}, C_q) \le \eta$ **then**
Return $C_q$
**End if**
**End for**
Return $C_q$
**End**

*Рис. 6. Псевдокод рекурсивной k-средней кластеризации*
*Fig. 6. Pseudocode of Recursive k-means Clustering*

## *4. Обсуждение результатов*

В этом разделе будет проведена сравнительная оценка производительности известного метода реструктуризации ООО-системы без нейронной сети (DOOR) и предлагаемого метода, в котором используется нейронная сеть (DOOR_NN). Оценка проводится сравнением затрат на организацию связей по совокупности кластеров. В системе моделирования MATLAB имеется возможность провести симуляцию такой реструктуризации. Симулятор включает дружественный пользовательский интерфейс, позволяющий пользователю указывать системные узлы и ребра, а также строить граф зависимостей.

*Табл. 1. Результаты моделирования для предлагаемой распределенной объектно-ориентированной реструктуризации с нейронной сетью и без нее.*
*Table 1. Simulation Results for the Proposed DOOR with Neural Network and the Existing technique without Neuron Network.*

| Число классов | Число кластеров | | Затраты на взаимодействие | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | Кластеризация | | Группирование | | Сопоставление с узлами сети | |
| | DOOR | DOOR_NN | RGC | CDBC | k-Partitioning | k-medoids | Двойная k-кластеризация | Рекурсивный k-средних |
| 51 | 6 | 4 | 2045 | 1916 | 1899 | 1714 | 1680 | 1540 |
| 62 | 7 | 6 | 2632 | 2243 | 2160 | 2095 | 2124 | 2024 |
| 79 | 9 | 7 | 2600 | 2415 | 2185 | 2060 | 2097 | 1988 |
| 107 | 11 | 9 | 3196 | 2818 | 3009 | 2698 | 2483 | 2184 |
| 153 | 15 | 13 | 4090 | 3825 | 3784 | 3515 | 3143 | 3005 |

В табл. 1 приведены результаты моделирования предложенной распределенной объектно-ориентированной реструктуризации с нейронной сетью (DOOR_NN) и известного метода без нейронной сети (DOOR). Предлагаемый метод DOOR_NN по всем позициям выигрывает у существующей стратегии DOOR, ориентированной на количество кластеров, кластеризацию, группировку и отображение. Для кластеризации в стратегии DOOR используется метод рекурсивной графовой кластеризации (Recursive Graph Clustering, RGC), а предложенный новый подход базируется на CDBC. Для группировки кластеров предложенная стратегия использует подход k-medoids, в то время как существующая процедура использует подход k-секционирования (k-Partitioning). Для отображения кластеров на серверные узлы в предложенной стратегит использовалась рекурсивная кластеризация k-средних, в то время как в существующем методе используется подход двойной k-кластеризации (Double k). Для любого количества классов и кластеров предлагаемая процедура с нейронной сетью демонстрирует превосходство по производительности по сравнению с существующим подходом без нейронной сети.

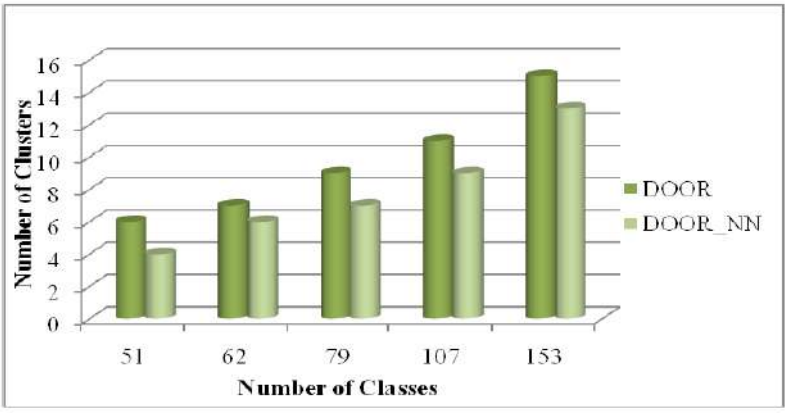


*Рис. 7. Анализ производительности предложенного метода в зависимости от числа кластеров*
*Fig. 7. Performance analysis of the proposed technique based on the number of clusters*

На рис. 7 анализируется предлагаемая производительность DOOR_NN и эффективность существующей процедуры DOOR, сконцентрированной на количестве кластеров, сформированных в процессе кластеризации. Он демонстрирует результаты моделирования с четырьмя узлами. Он также показывает совокупное количество кластеров, сформированных для заданного количества классов. Здесь можно безошибочно заметить, что предлагаемый метод DOOR_NN производит меньшее количество кластеров, чем существующая стратегия, что показывает большую эффективность предложенной стратегии.

## 4.1 Кластеризация

На рис. 8 иллюстрируется сравнение затрат на установление связей между кластерами для предложенной процедуры кластеризации на основе CDBC и для существующего метода RGC. По оси X на рис. 8 отмечается количество построенных кластеров, а по оси Y – затраты на установление связи в единицах времени, учитывающих расположение классов в различных узлах сети. Видно, что для четырех кластеров расходы на связь примерно одинаковы для процедуры RGC и предлагаемого метода CDBC. Однако по мере увеличения количества кластеров расходы на связь увеличиваются, при этом наблюдается различие в стоимости для существующей и предлагаемой стратегий. Предложенный метод CDBC демонстрирует наименьшие затраты при любом количестве кластеров.

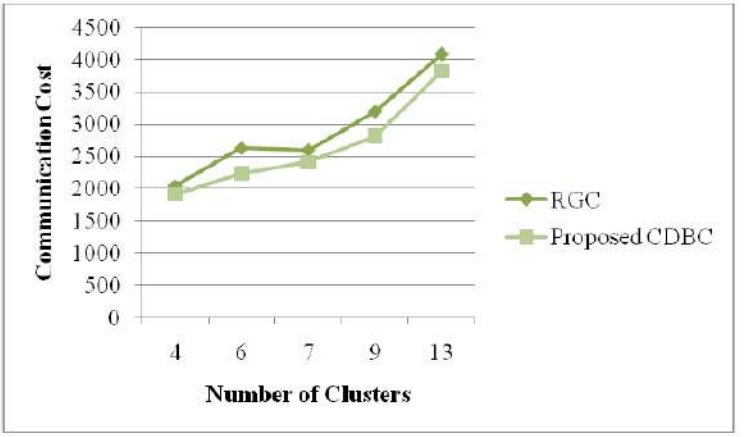


*Рис. 8. Анализ эффективности кластеризации в терминах затрат на взаимодействие для известного метода RGC и предлагаемого метода CDBC*

*Fig. 8. Performance analysis of Clustering in terms of Communication Cost for the existing RGC and the proposed CDBC*


*Рис. 9 Анализ эффективности группирования в терминах затрат на взаимодействие для известного метода k-секционирования и предлагаемого метода k-medoids*

*Fig. 9 Performance analysis of Grouping in terms of Communication Cost for the existing k-Partitioning and the proposed k-medoids*

## 4.2 Группировка

На рис. 9 иллюстрируется сравнение затрат на группирование кластеров для предложенного к использованию метода кластеризации k-medoids и часто используемой

стратегии кластеризации k-секционирования. Для проведения группирования кластеров предлагается использовать метод кластеризации k-medoids. При числе кластеров 6 и 7 затраты на связь примерно одинаковы как для метода k-секционирования, так и для предлагаемого метода k-medoids. Однако сравнение стоимости обоих методов показывает, что предлагаемый метод k-medoids имеет более низкую стоимость установления связей при любом количестве кластеров.

## 4.3 Отображение

Рис. 10 иллюстрирует сравнение затрат на установление связей при использовании для распределения кластеров по узлам сети предлагаемой системы рекурсивной кластеризации k-средних и часто используемой методики двойной k-кластеризации.

Отображение кластеров на узлы предлагается проводить на основе стратегии рекурсивной кластеризации k-средних. Разница в стоимости связи для предлагаемой и существующей систем особенно хорошо видна, когда количество кластеров равнялось 4, 6 и 7. Наибольшее различие между методами выявлено для девяти кластеров.


*Рис. 10. Анализ эффективности отображение в терминах затрат на взаимодействие для известного метода двойной k-кластеризации и предложенного рекурсивного метода k-средних*
*Fig. 10. Performance analysis of Mapping in terms of Communication Cost for the existing Double k-Clustering and the proposed Recursive k-means*

## *5. Заключение*

Использование методов объектно-ориентированного программирования позволяет заметно упростить создание приложений, предоставляя передовую платформу для разработки приложений. Реструктуризация программного обеспечения (ООО-систем) осуществляется с помощью предложенного адаптивного метода с использование нейронной сети, который способен еще больше повысить производительность системы. Благодаря использованию этого метода, можно уменьшить совокупность кластеров, которая первоначально возникает при обучении нейронной сети, и, следовательно, уменьшить совокупность выделенных ресурсов.

Производительность существующего метода реструктуризации распределенной объектно-ориентированной программной системы (распределения кластеров классов системы по узлам сети) без нейронной сети (DOOR) и предлагаемого метода реструктуризации с использованием нейронной сети (DOOR_NN) оценивается с помощью метрик производительности, например, количества кластеров и стоимости

связи. Построение кластеров с использованием нейронной сети значительно уменьшает количество кластеров, генерируемых во время кластеризации, и, следовательно, снижает стоимость связи. Результаты моделирования показали, что предлагаемая работа дает более хорошие результаты по сравнению с существующими методами.

## References


[1]. Wang J., Ai J., Yang Y., Su W. Identifying key classes of object-oriented software based on software complex network. In Proc. of the 2nd International Conference on System Reliability and Safety, 2017, pp. 444-449.

[2]. Tarwani S., Chug A. Prioritization of code restructuring for severely affected classes under release time constraints. In Proc. of the 1st India International Conference on Information Processing, 2016, pp. 1-6.

[3]. Boucher A., Badri M. Predicting Fault-Prone Classes in Object-Oriented Software: An Adaptation of an Unsupervised Hybrid SOM Algorithm. In Proc. of the IEEE International Conference on Software Quality, Reliability and Security, 2017, pp. 306-317.

[4]. Rajdev U., Kaur A. Automatic detection of bad smells from excel sheets and refactor for performance improvement. In Proc. of the International Conference on Inventive Computation Technologies, vol. 2, 2016, pp. 1-8.

[5]. Kaya M., Fawcett J.W. A new cohesion metric and restructuring technique for object oriented paradigm. In Proc. of the IEEE 36th Annual Computer Software and Applications Conference Workshops, 2012, pp. 296-301.

[6]. Mourad B., Badri L., Hachemane O., Ouellet A. Exploring the Impact of Clone Refactoring on Test Code Size in Object-Oriented Software. In Proc. of the 16th IEEE International Conference on Machine Learning and Applications, 2017, pp. 586-592.

[7]. Amirat A., Bouchouk A., Yeslem M.O., Gasmallah N. Refactor Software architecture using graph transformation approach. In Proc. of the Second International Conference on Innovative Computing Technology, 2012, pp. 117-122.

[8]. Bhatti M.U., Ducasse S., Huchard M. Reconsidering classes in procedural object-oriented code. In Proc. of the 15th Working Conference on Reverse Engineering, 2008, pp. 257-266.

[9]. Liu H., Li G., Ma Z.Y., Shao W.Z. Conflict-aware schedule of software refactorings. IET software, vol. 2, issue 5, 2008, pp. 446-460.

[10]. Nongpong K. Feature envy factor: A metric for automatic feature envy detection. In Proc. of the 7th International Conference on Knowledge and Smart Technology, 2015, pp. 7-12.

[11]. Tomyim J., Pohthong A. Requirements change management based on object-oriented software engineering with unified modeling language. In Proc. of the 7th IEEE International Conference on Software Engineering and Service Science, 2016, pp. 7-10.

[12]. Hamad S.H., Ammar R.A., Khalifa M.E., Fergany T. Randomized Algorithms for Mapping Clustered Object-Oriented Software onto Distributed Architectures. In Proc. of the 7th IEEE International Symposium on Signal Processing and Information Technology, 2018, pp. 426-431.

[13]. Sugandhi R., Srivastava P., Sanyasi A., Awasthi L.M., Parmar V., Makadia K., Patel I., Shah S. Implementation of object oriented software engineering on LabVIEW graphical design framework for data acquisition in large volume plasma device. In Proc. of the 7th International Conference on Cloud Computing, Data Science & Engineering, 2017, pp. 798-803.

[14]. Faheem M.T., Ammar R.A., Sarhan A.M., Ragab H.A.M. A hybrid algorithm for restructuring distributed Object-oriented software. In Proc. of the International Symposium on Signal Processing and Information Technology, 2010, pp. 202-208.

[15]. Cosma D.C. Reverse engineering object-oriented distributed systems. In Proc. of the IEEE International Conference on Software Maintenance, 2010, pp. 1-6.

[16]. Gu A., Zhou X., Li Z., Li Q., Li L. Measuring Object-Oriented Class Cohesion Based on Complex Networks. Arabian Journal for Science and Engineering, vol. 42, no. 8, 2017, pp. 3551-3561.

[17]. Ajienka N., Capiluppi A., Counsell S. An empirical study on the interplay between semantic coupling and co-change of software classes. Empirical Software Engineering, 2017. pp. 1-35.

[18]. Kumar N., Dadhich R., Shastri A. MAQM: a generic object-oriented framework to build quality models for Web-based applications. International Journal of System Assurance Engineering and Management, vol. 8, no. 2, pp. 2017, pp. 716-729.

[19]. Parashar A., Chhabra J.K. Mining software change data stream to predict changeability of classes of object-oriented software system. Evolving Systems, vol. 7, no. 2, 2016, pp. 117-128.
[20]. Nucci D.D., Palomba F., Rosa G.D., Bavota G., Oliveto R., Lucia A.D. A developer centered bug prediction model, IEEE Transactions on Software Engineering, vol. 44, no. 1, 2018, pp. 5-24.
[21]. Aslam W., Ijaz F. A Quantitative Framework for Task Allocation in Distributed Agile Software Development. IEEE Access, vol. 6, 2018, pp. 15380-15390.

## Информация об авторе / Information about author

Ахмед ХАН работает преподавателем на факультете компьютерных наук. Его научные интересы включают искусственный интеллект, параллельные вычисления и теорию информации.

Ahmed KHAN is a lecture at the Department of Computer Science. His research interests include artificial intelligence, parallel processing, and information theory.

**DOI: 10.15514/ISPRAS-2019-31(5)-9**

# Cross-lingual similar document retrieval methods

*D.V. Zubarev, ORCID:* 0000-0*002-9687-6650 <zubarev@isa.ru>*
*I.V. Sochenkov, ORCID: 0000-0003-3113-3765 <sochenkov@isa.ru>*

*Federal Research Center «Computer Science and Control» of Russian Academy of Sciences, 44-2 Vavilov Str., Moscow 119333, Russia*

**Abstract.** In this paper, we compare different methods for cross-lingual similar document retrieval. We focus on Russian-English language pair. We compare well-known methods like Cross Lingual Explicit Semantic Analysis (CL-ESA) with methods based on cross-lingual embeddings. We use approximate nearest neighbor (ANN) search to retrieve documents based entirely on distances between learned document embeddings. Also we employ a more traditional approach with usage of inverted index, with extra step of mapping top keywords from one language to other with the help of cross-lingual word embeddings. We use Russian-English aligned Wikipedia articles to evaluate all approaches. Conducted experiments show that an approach with inverted index achieves better performance in terms of recall and MAP than other methods.

**Keywords:** cross-lingual document retrieval; cross-lingual plagiarism detection; cross-lingual word embeddings.

**For citation:** Zubarev D.V., Sochenkov I.V. Cross-lingual similar document retrieval methods. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019, pp. 127-136. DOI: 10.15514/ISPRAS-2019-31(5)-9

**Acknowledgements.** This study was funded by RFBR according to the research project № 18-37-20017. The reported research is also partially funded by the project “Text mining tools for big data” as a part of the program supporting Technical Leadership Centers of the National Technological Initiative “Center for Big Data Storage and Processing” at the Moscow State University (Agreement with Fund supporting the NTI-projects No. 13/1251/2018 11.12.2018).

## Методы кросс-языкового поиска похожих документов

*Д.В. Зубарев, ORCID: 0000-0002-9687-6650 <zubarev@isa.ru>*
*И.В. Соченков, ORCID: 0000-0003-3113-3765 <sochenkov@isa.ru>*
*Федеральный исследовательский центр «Информатика и управление» РАН, 119333, Россия, г. Москва, ул. Вавилова, д. 44, к. 2*

**Аннотация.** В этой статье сравниваются различные методы кросс-языкового поиска похожих документов. Для сравнения используется русско-английская языковая пара. Сравниваются известные методы, такие как CL-ESA, с методами, основанными на кросс-языковых эмбеддингах. Для поиска документов используется приближенный поиск ближайшего соседа (ANN), использующий расстояния между векторами, представляющими документы. Также применяется более традиционный подход с использованием инвертированного индекса, с дополнительным шагом: отображение ключевых слов с одного языка на другой с помощью кросс-языковых эмбеддингов. Для экспериментальной оценки всех методов используются русские статьи из Википедии, которые имеют аналоги в англоязычной версии. Проведенные эксперименты показывают, что подход с инвертированным индексом показывает лучшие результаты по двум метрикам: полнота и средняя точность (MAP).

**Ключевые слова:** кросс-языковой поиск похожих документов; кросс-языковой поиск заимствований; кросс-языковые эмбеддинги

**Для цитирования:** Зубарев Д.В., Соченков И.В. Методы кросс-языкового поиска похожих документов. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 127-136 (на английском языке). DOI: 10.15514/ISPRAS-2019-31(5)-9

**Благодарности.** Работа выполнена при поддержке гранта РФФИ № 18-37-20017. Исследование выполнено также при частичной финансовой поддержке проекта "Средства интеллектуального анализа больших массивов текстов" в рамках программы Центров компетенций Национальной технологической инициативы на базе Московского государственного университета им. М.В. Ломоносова (соглашение о финансовой поддержке проектов НТИ № 13/1251/2018 от 11.12.2018)

## *1. Introduction*

Document retrieval from a large collection of texts is important information retrieval problem. This problem is extensively studied for short queries, such as user queries to search engines.

The document retrieval with texts as queries impose some difficulties, among them inability to capture the main ideas and topics from the long text. The problem becomes even harder when we enter the field of cross-lingual document retrieval. Some tasks require to use a text (possibly long) as query to retrieve documents that are somehow similar to it. One of these tasks is plagiarism detection that is divided into two stages: source retrieval and text alignment.

- On the source retrieval stage for a given suspicious document, we need to find all sources of probable text reuse in a large collection of texts. For this task, a source is a whole text, without details of what parts of this document were plagiarized. Typically, we get a large set of documents (around 500 or more) as a result of this stage.
- On the text alignment stage: we compare the suspicious document to each candidate to detect all reused fragments, and identify its boundaries [1-4].

In this work, we study only the first task. The same stages are valid for cross-lingual plagiarism detection. Given a query document in one language the goal is to find the most similar documents from the collection in another language.

## *2. Related work*

Some works were recently devoted to the monolingual document retrieval for long texts. In [5], the authors introduce a siamese multi-depth attention-based hierarchical recurrent neural network that learns the long text semantics. They conducted multiple experiments including retrieval of similar Wikipedia articles. In [6], the authors try to employ standard approximate nearest neighbor (ANN) search instead of the usual discrete inverted index, for retrieving documents. They learned similarity function and showed that it can improve performance on two similar-question retrieval tasks. However, using the custom similarity functions makes impossible to employ existing frameworks for ANN, consequently they used exact search in experiments.

In [7], a framework is introduced for monolingual and cross-lingual information retrieval based on cross-lingual word embeddings. They represent user queries and documents as averaged embeddings of words and employ exact search to find similar documents for a given query. The overview of different approaches for cross-lingual source retrieval is presented in [8] and [9]. Also, there made an evaluation and a detailed comparison of some featured methods.

In [10], NMT (neural machine translation) is used to translate a query document to other language. They solve source retrieval task by employing shingles (overlapping word N-grams) method. They use word-class shingles, instead of word shingles, where each word is substituted by the label of the class it belongs to. To obtain word classes they apply agglomerative clustering on word embeddings learned from English Wikipedia.

The work [11] describes a training of word embeddings on comparable monolingual corpora and learning the optimal linear transformation of vectors from one language to another (there were

used Russian and Ukrainian academic texts). Also there were discussed usage of those embeddings in source retrieval and text alignment subtasks. This work focuses on comparison of retrieval-based approaches with ANN approach for distant language pair.

## *3. Document retrieval methods*

In this section, we describe various methods that we used for document retrieval.

### 3.1 Preprocessing

On a preprocessing stage, we split each sentence into tokens, lemmatize tokens and parse texts. We use AOT for the Russian language and Udpipe[1] [12] for English language. Besides, we removed words with non-important part of speech: conjunction, pronoun, preposition, etc., and common stop-words (be, являться, etc.).

### 3.2 Cross-lingual embeddings

We train cross-lingual word embeddings for a Russian-English pair on parallel sentences available on the Opus site [13] namely:

- News Commentary;
- TED Talks 2013;
- MultiUN (first 2 million sentences);
- Wiki;
- JW300;
- QED;
- Tatoeba.

We extend this corpus with sentences from the Yandex Parallel corpus[2] [14].

All parallel sentences are preprocessed. After that, all pairs that have a difference in the size of more than 10 words are filtered out. We use syntactic phrases up to 4 words in length to enrich the vocabulary. We take only those phrases (noun phrases and some prepositional phrases for the English language) that are common for the corpus (>10 occurrences). We duplicate one sentence multiple times if there are some overlapping phrases. For example, from the sentence with the phrase «Russian presidential election ...» will be generated three variations with different phrases:

- «Russian_presidential_election ... »;
- «Russian_election presidential_election ... »;
- «Russian presidential election ... ».

Finally, we assembled a corpus of more than 5.1 million sentences (more than 10 million sentences with phrases variations). The dictionary size was around 680 000 words/phrases.

We apply two different methods for learning cross-lingual embeddings [15].

First, we learn monolingual embeddings for each language. We use word2vec skip-gram model [16] with the following parameters: dimensionality of embeddings was 300, a window size of 10 words, the minimal corpus frequency of 10, negative sampling with 10 samples, no down-sampling, 20 iterations over the corpus. Then we use vecmap [17, 18] framework to learn a transformation matrix that maps representations in one language to the representations of the

[1] english-ewt-ud-2.4-190531 model

[2] Англо-русский параллельный корпус: https://translate.yandex.ru/corpus?lang=en

other language. We use 20 000 random word pairs from the bilingual dictionary of MUSE project[3] [19] as the training data.

Second, we apply the method proposed in [20], designed for learning bilingual word embeddings from a non-parallel document-aligned corpus, but it can be used for learning on parallel sentences too. We assume that the structures of the two sentences are similar. Words are inserted into the pseudo-bilingual sentence relying on the order in which they appear in their monolingual sentences and based on length ratio of two sentences. For example, if we were given two sentences: «Мама мыла раму» and «Mother washed beautiful frame», the result of their merging is «мама mother мыла washed раму beautiful frame». Since we removed auxiliary words from sentences, we assume that corresponding Russian and English words are in the same context window. It would not be the case if there were a different word order, so we experimented with different window sizes and chose size == 10. After that, the word2vec skip-gram model is used on the resulting bilingual corpus. We use gensim word2vec implementation with those parameters: dimensionality of embeddings was 300, a window size of 10 words, the minimal corpus frequency of 10, negative sampling with 10 samples, no down-sampling, 20 iterations over the corpus.

## 3.3 Retrieval-based approach

We use a custom implementation of inverted index [21], which maps each word to a list of documents in which it appears along with weight (e.g. TF) that represents the strength of association of this word with a document. Along with words, we index syntactic phrases up to 4 words, which occur in a document more than once.

At query time, we extract the top words/phrases from the query document according to some weighting scheme. Then we map each keyword to N other language keywords with cross-lingual embeddings. We precompute the most similar words for each word in our vocabulary to speed up this operation. We preserve the weights of keywords from the original top. The searcher iterates over the top keywords, retrieves corresponding documents from the inverted index, and merges them into weighted vectors of keywords that represent the other documents. Then we compare the query vector with all other vectors. It should be noted that comparison is asymmetrical since vectors of other documents consist only of words from the query vector. Although it is not the most accurate representation of these documents, the comparison is very efficient, and retrieval performance (recall) is not affected much by that. To compute the similarity score between vectors we employ some similarity measure (e.g. cosine similarity).

## 3.4 Approximate nearest neighbor search (ANN)

In this approach, we represent each document as a dense vector. It is done by averaging vectors of the top K keywords of the document. After that, we index all vectors with ANN index. At query time, the given document is transformed into the vector representation, and the approximate nearest neighbor search is employed to retrieve the most similar documents.

## 3.5 Explicit semantic analysis (ESA)

We implemented CL-ESA method described in [9] and firstly introduced for solving monolingual semantic relatedness task in [22]. This method represents the document as a weighted vector of concepts. Concepts are defined by Wikipedia articles. In the original work, the authors used all English Wikipedia articles as concepts. We selected around 800 000 English articles that are aligned with Russian Wikipedia articles (articles that identified as comparable across languages by the Wikipedia community). For a given document *D* the weight of a concept

[3] A library for Multilingual Unsupervised or Supervised word Embeddings, https://github.com/facebookresearch/MUSE

$C$ is defined as cosine similarity between top M keywords of $D$ and matched keywords of an article that is linked with the concept $C$:

$$\frac{\sum_{w_i \in D} v_i \cdot c_i}{\sqrt{\sum_{w_i \in D} v_i^2} \sqrt{\sum_{w_i \in D} c_i^2}}$$

where $v_i$ is the weight of a word $w_i$ for $D$ (e.g. TF-IDF), $c_i$ is the weight of a word $w_i$ for a Wikipedia article linked with the concept (e.g. TF-IDF).

We precomputed vector of concepts for each document in text collection and stored them with the same inverted index implementation that was used for the retrieval-based approach.

At query time, the query document is converted to a vector of weighted concepts, i.e., identificators of Wikipedia articles. Then those identificators are mapped to articles in other language, and similar documents are retrieved via search in the inverted index.

## 4. Dataset

We use Russian-English aligned Wikipedia articles as a dataset for evaluation of retrieval methods (Wikipedia dump of June 2019). We exclude all articles, which title starts with words "List of", which size in symbols is less than 800, and which number of sentences is less than 10.

Then we divide all remaining pairs of articles into two groups and each group into five bins by the size of a Russian article in sentences:

- comparable by size: those articles that satisfy the following requirement:

$$|len(a_{\text{ru}}) - len(a_{\text{en}})| < min(len(a_{\text{ru}}), len(a_{\text{en}}))/4$$

- non-comparable by size: Those articles that satisfy the following requirement:

$$|len(a_{\text{ru}}) - len(a_{\text{en}})| > min(len(a_{\text{ru}}), len(a_{\text{en}}))$$

*Table 1: Statistic of comparable by size articles*

| Size in ru sents | count | Mean sife of ru texts | Mean size of en texts |
|---|---|---|---|
| (9, 50] | 62291 | 2560.34 | 2626.16 |
| (50, 100] | 20012 | 5878.33 | 5989.66 |
| (100, 200] | 9163 | 11100.2 | 11301.2 |
| (200, 400] | 3526 | 21275 | 21693.2 |
| (400, 1000] | 1628 | 43478.6 | 44023.1 |

*Table 2: Statistic of non-comparable by size articles*

| Size in ru sents | count | Mean sife of ru texts | Mean size of en texts |
|---|---|---|---|
| (9, 50] | 170902 | 2336.02 | 11471.6 |
| (50, 100] | 42958 | 6076.27 | 17771.8 |
| (100, 200] | 22491 | 11519.4 | 22309.7 |
| (200, 400] | 9318 | 21425.2 | 26847.4 |
| (400, 1000] | 3517 | 42744.7 | 26895 |

Then we sampled 100 documents from each group. That gives us a dataset that contains 1000 document pairs[4].

[4] http://nlp.isa.ru/ru-en-src-retr-dataset/

### 4.1 Indexing of Wikipedia

We indexed all articles from English (5.8M) and Russian (1.5M) Wikipedia dumps (June 2019).

#### 4.1.1 Retrieval-based approach

We use TF-IDF weighting scheme with $\log_{len(D)+1}(Cnt(w_i)+1)$ as TF weight for word $w_i$ from a document $D$, and $max(0, \log_{10}(N - w_{\text{cnt}} + 0.5)/(w_{\text{cnt}} + 0.5))$ as IDF, where $N$ is total amount of documents in a collection.

#### 4.1.2 Approximate nearest neighbor search

We take top keywords with weight > 0.05 and average embeddings of those words. We use Faiss IVFFlatIndex [23] for indexing document embeddings with the following parameters: number of centroids - $4 * \sqrt{|V|}$, where $V$ set of all vectors that we need to index, training set size – $5 \times min\_points\_per\_centroid \times num\_centroids$, where $min\_points\_per\_centroid$ is equal 39 by default, nprobe – 16, compression – SQfp16. Our experiments showed that these parameters result in efficient search time and search precision greater than 90%.

#### 4.1.3 ESA

When precomputing concept vectors for ESA method, we used 200 top keywords (with weight > 0.05) of a document to compute weights of concepts. We kept the maximum 1200 concepts with the largest weight per document. Since we build vectors of Wikipedia articles using Wikipedia articles as concepts, we excluded a concept that represents the same article from the vectors.

## *5. Evaluation Results*

We used grid search for parameters tuning on 400 documents that were sampled independently of the testing data. We performed a search on all 1000 documents using various methods, retrieved the most similar 600 documents and measured standard metrics: Recall, MAP. We use the following abbreviation in the table 3 and below:

- RBA – Retrieval-based approach;
- EMB – Embeddings that were used: BIL - embeddings built on bilingual corpus, MAP - embeddings mapped via Vecmap framework;
- MP – Maximum phrase size (1-4), if 1 the keywords may only contain single words;
- N – Number of similar words in other language that were taken for each word when mapping keywords (1 if not specified explicitly);
- MTS – Number of keywords in other language (mapped top size) (100 if not specified explicitly);
- SK – Similarity score: cosine (cos) or hamming (ham) similarity measures (cos if not specified explicitly);
- ANN – Approximate nearest neighbor search;
- DIM – Dimensionality of embeddings (300 if not specified explicitly);
- K – Document is an average of vectors of the top K keywords;
- ESA – Explicit semantic analysis;
- CTOP – Number of concepts to use for retrieval.

The Table 3 displays the evaluation results obtained on the wiki dataset.

*Table 3. Evaluation results*

| **Method** | **Rec@1** | **Rec@10** | **Rec@20** | **Rec** | **MAP** |
|---|---|---|---|---|---|
| RBA (EMB=BIL,MP=1) | 0.415 | 0.622 | 0.66 | 0.831 | 0.48 |
| RBA (EMB=BIL,MP=2) | **0.418** | **0.632** | **0.67** | 0.843 | **0.49** |
| RBA (EMB=BIL,MP=4) | 0.415 | **0.635** | **0.67** | 0.845 | **0.49** |
| RBA (EMB=BIL,DIM=600,MP=4) | **0.428** | 0.629 | **0.671** | **0.856** | **0.5** |
| RBA (EMB=BIL,MP=4,SK=ham) | 0.387 | 0.611 | 0.661 | **0.849** | 0.467 |
| RBA (EMB=MAP,MP=4) | 0.263 | 0.478 | 0.533 | 0.767 | 0.336 |
| ANN (EMB=BIL,MP=2,K=25) | 0.313 | 0.508 | 0.548 | 0.715 | 0.379 |
| ANN (EMB=BIL,MP=2,K=50) | 0.337 | 0.508 | 0.548 | 0.728 | 0.398 |
| ANN (EMB=BIL,MP=2,K=100) | 0.266 | 0.433 | 0.475 | 0.689 | 0.323 |
| ANN (EMB=BIL,DIM=600,MP=2,K=50) | 0.374 | 0.527 | 0.577 | 0.724 | 0.433 |
| ANN (EMB=MAP,MP=2,K=50) | 0.197 | 0.36 | 0.426 | 0.665 | 0.254 |
| ESA (CTOP=200,SK=cos) | 0.254 | 0.453 | 0.501 | 0.833 | 0.318 |

The results show that the retrieval-based approach is better in terms of Recall and Map than other methods. The embeddings, built on the bilingual corpus (EMB=BIL), give better results for this task than embeddings obtained via mapping (EMB=MAP). The results of experiments show that syntactic phrases give no significant boost in performance for RBA and ANN approaches. Doubling the number of components of embeddings from 300 to 600 results in better ranking for ANN aproach, but almost has no effect for RBA. ESA shows good recall, but ranking of found documents is worse than for the RBA and ANN methods.

It should be pointed out that the performance of the methods differs significantly depending on the size of the documents (table 4).

*Table 4: RBA (EMB=BIL,MP=4, MTS=100/50) Metrics per each size group*

| **No** | **size in ru sents** | **comparable by size?** | **MAP (MTS=100)** | **Rec (MTS=100)** | **MAP (MTS=50)** | **Rec (MTS=50)** |
|---|---|---|---|---|---|---|
| 1 | (9, 50] | False | 0.346 | 0.82 | 0.346 | 0.82 |
| 2 | (9, 50] | True | 0.338 | 0.65 | 0.338 | 0.65 |
| 3 | (50, 100] | False | 0.419 | 0.79 | 0.419 | 0.8 |
| 4 | (50, 100] | True | 0.44 | 0.88 | 0.445 | 0.88 |
| 5 | (100, 200] | False | 0.461 | 0.81 | 0.453 | 0.82 |
| 6 | (100, 200] | True | 0.542 | 0.88 | 0.535 | 0.9 |
| 7 | (200, 400] | False | 0.451 | 0.79 | 0.473 | 0.8 |
| 8 | (200, 400] | True | 0.730 | 0.98 | 0.742 | 0.97 |
| 9 | (400, 1000] | False | 0.306 | 0.85 | 0.341 | 0.81 |
| 10 | (400, 1000] | True | 0.87 | 1 | 0.871 | 1 |

RBA method works better with long texts that are comparable by size. Short texts (groups 1,2) are likely to have some specific out-of-vocabulary lexis, and remaining words do not help to retrieve the article in other language and rank it highly. The lowest MAP is for the combination 9, i.e. the non-comparable by size long texts, whereas the best MAP is for the longest texts also, but this time for comparable ones (group 10). It can be seen from table 2 that Russian texts from this group are longer than English texts the factor of two. These articles may be devoted to Russian concepts that have short descriptions in English. In this case, similar articles with longer

texts have more chances to share lexis than shorter articles. Therefore, shorter texts are lower in the rank list.

Similar behavior is observed for other methods too. For example, table 5 presents the comparison of results for different K (50, 25) for ANN method.

*Table 5. ANN (EMB=BIL, MP=2, K=50/25) Metrics per each size group*

| No | Size in ru sents | comparable by size? | MAP (K=50) | Rec (K=50) | MAP (K=25) | Rec (K=25) |
|---|---|---|---|---|---|---|
| 1 | (9, 50] | False | 0.228 | 0.59 | 0.192 | 0.55 |
| 2 | (9, 50] | True | 0.226 | 0.61 | 0.351 | 0.69 |
| 3 | (50, 100] | False | 0.237 | 0.53 | 0.214 | 0.52 |
| 4 | (50, 100] | True | 0.482 | 0.72 | 0.433 | 0.69 |
| 5 | (100, 200] | False | 0.334 | 0.72 | 0.352 | 0.74 |
| 6 | (100, 200] | True | 0.562 | 0.81 | 0.472 | 0.79 |
| 7 | (200, 400] | False | 0.328 | 0.73 | 0.357 | 0.71 |
| 8 | (200, 400] | True | 0.56 | 0.87 | 0.507 | 0.81 |
| 9 | (400, 1000] | False | 0.272 | 0.74 | 0.229 | 0.74 |
| 10 | (400, 1000] | True | 0.754 | 0.96 | 0.688 | 0.91 |

There are some groups where performance is better with a lesser amount of keywords, e.g., 2, 5 (recall and MAP), and 7 (MAP). This result suggests that our strategy of selecting keywords (select up to N words with weight > X) does not work well for all cases.

## *6. Conclusion*

In this article, we compared various methods for cross-lingual retrieval of similar documents. We employed classical inverted indexes combined with cross-lingual embeddings and pure continuous retrieval using ANN. For this task and our dataset, the best result was shown by retrieval-based approach. It achieves the best recall and MAP scores with long comparable by size texts. As future work, we need to focus on improving performance for non-comparable by size texts, since now its ranking is far from the good. Dealing with OOV is another important issue. One way to solve it is to employ subword vector representation to encode the OOV-words [24]. Another way is to extending vocabulary from different comparable corpora: scientific papers, patents, etc. containing a lot of special lexis and terms. One of the possible solutions is to use the system for translated plagiarism detection to extract parallel sentences from comparable corpora [25].

## References / Список литературы

[1]. Romanov A., Kuznetsova R., Bakhteev O., Khritankov A. Machine-Translated Text Detection in a Collection of Russian Scientific Papers. In Proc. of the Annual International Conference “Dialogue”, 2016.

[2]. Zubarev D.V., Sochenkov I.V. Cross-language text alignment for plagiarism detection based on contextual and context-free models. In Proc. of the Annual International Conference “Dialogue” 2019, v. 1, pp. 799-810.

[3]. Ferrero J., Agnes F., Besacier L., Schwab D. Using Word Embedding for Cross-language Plagiarism Detection. arXiv:1702.03082, 2017.

[4]. Franco-Salvador M., Gupta P., Rosso P., Banchs R.E. Cross-language plagiarism detection over continuous space and knowledge graph-based representations of language. Knowledge-based systems, vol. 111, 2016, pp. 87-99.

[5]. Jiang J.Y., Zhang M., Li C., Bendersky M., Golbandi N., Najork M. Semantic Text Matching for Long-Form Documents. In Proc. of the World Wide Web Conference, 2019, pp. 795-806.

[6]. Gillick D., Presta A., Tomar G.S. End-to-End Retrieval in Continuous Space. arXiv:1811.08008, 2018.
[7]. Vulić I., Moens M.F. Monolingual and cross-lingual information retrieval models based on (bilingual) word embeddings. In Proc. of the 38th international ACM SIGIR conference on research and development in information retrieval, 2015, pp. 363-372.
[8]. Barrón-Cedeño A., Gupta P., Rosso P. Methods for cross-language plagiarism detection. Knowledge-Based Systems, vol. 50, 2013, pp. 211-217.
[9]. Potthast M., Barrón-Cedeño A., Stein B., Rosso P. Cross-language plagiarism detection. Language Resources and Evaluation, vol. 45, issue 1, 2011, pp. 45-62.
[10]. Bakhteev O., Ogaltsov A., Khazov A., Safin K., Kuznetsova R. CrossLang: the system of cross-lingual plagiarism detection. In Proc. of the KDD Workshop on Deep Learning for Education, 2019. Available at: https://truth-discovery-kdd2019.github.io/papers/crosslang.pdf, accessed 15.11.2019
[11]. Kutuzov A., Kopotev M., Sviridenko T., Ivanova L. Clustering Comparable Corpora of Russian and Ukrainian Academic Texts: Word Embeddings and Semantic Fingerprints. In Proc. of the Ninth Workshop on Building and Using Comparable Corpora, 2016, pp. 3-10
[12]. Straka M., Hajic J., Straková J. UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, pos tagging and parsing. In Proc. of the tenth international conference on language resources and evaluation, 2016, pp. 4290-4297.
[13]. Tiedemann J. Parallel Data, Tools and Interfaces in OPUS. In Proc. of the language resources and evaluation (LREC), 2012, pp. 2214-2218.
[14]. Antonova A., Misyurev A. Building a web-based parallel corpus and filtering out machine-translated text. In Proc. of the 4th Workshop on Building and Using Comparable Corpora: Comparable Corpora and the Web, 2011, pp. 136-144.
[15]. Ruder S., Vulić I., Søgaard A. A survey of cross-lingual word embedding models. Journal of Artificial Intelligence Research, vol. 65, issue 1, 2019, pp. 569-631.
[16]. Mikolov T., Sutskever I., Chen K., Corrado G.S., Dean J. Distributed representations of words and phrases and their compositionality. In Proc. of the 26th International Conference on Neural Information Processing Systems, 2013, vol. 2, pp. 3111-3119.
[17]. Artetxe M., Labaka G., Agirre E. Generalizing and improving bilingual word embedding mappings with a multi-step framework of linear transformations. In Proc. of the Thirty-Second AAAI Conference on Artificial Intelligence, 2018, pp. 5012-5019.
[18]. Glavas G., Litschko R., Ruder S., Vulic I. How to (Properly) Evaluate Cross-Lingual Word Embeddings: On Strong Baselines, Comparative Analyses, and Some Misconceptions. arXiv:1902.00508, 2019.
[19]. Conneau A., Lample G., Ranzato M. A., Denoyer L., Jégou H. Word translation without parallel data. arXiv:1710.04087, 2017.
[20]. Vulić I., Moens M.F. Bilingual word embeddings from non-parallel document-aligned data applied to bilingual lexicon induction. In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, 2015, vol. 2, pp. 719-725.
[21]. Sochenkov I.V., Zubarev D.V., Tikhomirov I.A. Exploratory patent search. Informatics and its Applications, vol. 12, issue 1, 2018, pp. 89-94 (in Russian). / Соченков И.В., Зубарев Д.В., Тихомиров И.А. Эксплоративный патентный поиск. Информатика и ее применения, том 12, вып. 1, 2018 г., стр. 89-94..
[22]. Gabrilovich E., Markovitch S. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In Proc. of the 20th international joint conference on Artifical intelligence, 2007, pp. 1606-1611.
[23]. Johnson J., Douze M., Jégou H. Billion-scale similarity search with GPUs. arXiv:1702.08734, 2017.
[24]. Bojanowski P., Grave E., Joulin A., Mikolov T. Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 2017, vol. 5, pp. 135-146.
[25]. Zweigenbaum P., Sharoff S., Rapp R. Overview of the third BUCC shared task: Spotting parallel sentences in comparable corpora. In Proc. of 11th Workshop on Building and Using Comparable Corpora, 2018, pp. 39-42.

## Информация об авторах / Information about authors

Денис Владимирович ЗУБАРЕВ – инженер-исследователь ФИЦ ИУ РАН. Сфера научных интересов: информационный поиск, компьютерная лингвистика, поиск текстовых заимствований, анализ больших массивов данных.

Denis Vladimirovich ZUBAREV – Engineer of FRC CSC RAS. Research interests: information retrieval, natural language processing, plagiarism detection, big data.

Илья Владимирович СОЧЕНКОВ – кандидат физико-математических наук, заведующий отделом «Интеллектуальные технологии и системы» ФИЦ ИУ РАН. Область научных интересов: интеллектуальные методы поиска и анализа информации, компьютерная лингвистика, компьютерный анализ естественного языка, машинное обучение, анализ больших массивов данных.

Ilya Vladimirovich SOCHENKOV – PhD, Head of the Department of Intelligent Technologies and Systems of FRC CSC RAS. Research interests: information retrieval, natural language processing, machine learning, big data.

**DOI: 10.15514/ISPRAS-2019-31(5)-10**



# Methods for News Items Popularity Estimation on Early Stages

[2]*A.A. Avetisyan, ORCID: 0000-0002-7066-6954 <a.a.avetisyan@ispras.ru>*
[1] *M.D. Drobyshevskiy, ORCID: 0000-0002-1639-9154 <drobyshevsky@ispras.ru>*
[1,2] *D.Yu. Turdakov, ORCID: 0000-0001-8745-0984 <turdakov@ispras.ru>*

[1] *Ivannikov 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, Russian Federation*

**Abstract.** Millions of news are distributed online every day. Tools for predicting the popularity of news stories are useful to ordinary people to discover important information before it becomes generally known. Also, such methods can be used to increase the effectiveness of advertising campaigns or to prevent the spread of fake news. One of the important features for predicting information spread is the structure of the influence graph. However, this feature is usually not available for news, because authors rarely post explicit links to information sources. We propose a method for predicting the most popular news in the information flow, which solves this problem by constructing a latent graph of influence. Computational experiments with two different datasets have confirmed that our model improves the precision and recall of forecasting the popularity of news stories.

**Keywords:** information diffusion; information cascades; networks of diffusion

**For citation:** Avetisyan A.A., Drobyshevskiy M.D., Turdakov D.Yu. Methods for News Items Popularity Estimation on Early Stages. Trudy ISP RAN/Proc. ISP RAS, 2019, vol. 31, issue 5, pp. 137–144. DOI: 10.15514/ISPRAS-2019-31(5)-10

## Методы оценки популярности новостных материалов на ранних стадиях

[2] *А.А. Аветисян, ORCID: 0000-0002-7066-6954 <a.a.avetisyan@ispras.ru>*
[1] *М.Д. Дробышевский, ORCID: 0000-0002-1639-9154 <drobyshevsky@ispras.ru>*
[1,2] *Д.Ю. Турдаков, ORCID: 0000-0001-8745-0984 <turdakov@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова 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*

**Аннотация.** Миллионы новостей распространяются онлайн каждый день. Инструменты для прогнозирования популярности новостных материалов полезны для простых людей, чтобы обнаружить важную информацию, прежде чем она станет общеизвестной. Также такие методы можно использовать для повышения эффективности рекламных кампаний или предотвращения распространения поддельных новостей. Одной из важных особенностей прогнозирования распространения информации является структура графа влияния. Однако обычно для новостей она неизвестна, поскольку авторы редко публикуют явные ссылки на источники информации. Мы предлагаем метод прогнозирования наиболее популярных новостей в информационном потоке,

который решает эту проблему путем построения скрытого графа влияния. Вычислительные эксперименты с двумя различными наборами данных подтвердили, что наша модель повышает точность и точность прогнозирования популярности новостных сообщений.

**Ключевые слова:** распространение информации; информационные каскады; сети распространения

**Для цитирования:** Аветисян А.А., Дробышевский М.Д., Турдаков Д.Ю. Методы оценки популярности новостных материалов на ранних стадиях. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 137-144 (на английском языке). DOI: 10.15514/ISPRAS-2019-31(5)-10

## *1. Introduction*

Prediction of the popularity of data streams can be used in various scenarios such as political campaigns, preventing the spread of fake news, viral advertising, etc. In recent works, four types of features are distinguished for information popularity prediction: temporal features, structural features, features of early adopters [1], and content features. Structural features are based on explicit graphs, therefore they are applied only for analysis of online social networks.

Most of news platforms do not have explicit links between each other. As a result, it becomes impossible to track how the interaction between different resources affects the popularity of the news. However, we can assume the existence of a hidden network of influence between news sources and try to reconstruct it.

Thereby, we emphasise the two directions of studying information propagation: 1) inferring an influence graph under the assumption that information is spread along its edges, and 2) prediction of news popularity based on information flow features.

We define information propagation for a particular message as a chain

$$c = \big((u_1;\, t_1;\, \Theta_1);\, (u_2;\, t_2;\, \Theta_2), \dots, (u_n;\, t_n;\, \Theta_n)\big),$$

where $u_i$ is the *i*-th disseminator who posted the message, $t_i$ is a corresponding publication time, and $\Theta_i$ is an information about the post. We will call such chains *cascades*. A cascade will be considered popular if it contains more messages than *n*% of other cascades in the flow. The early stage of a cascade is the time when a small fixed number *k* of disseminators posted the message. In this paper we solve the following problem: to predict at the early stage with $k = 5$ whether a cascade will become larger than 50% of the other cascades in the flow.

In this paper, we combine two directions of studying information propagation. We propose a feature-based model that predicts news popularity in the flow. In case we do not have social graph, we estimate the latent graph of influence and use its structural features which improve the precision and recall of prediction.

The main contribution of the paper is a method for estimating information popularity that reconstructs structural features when they are clearly not available and experimental confirmation of its efficiency on two datasets of different nature. We also showed that even if a social graph is available, it is possible to construct a much smaller graph of influence, which is an equally strong feature for predicting popularity.

The rest of the paper is organized as follows. In the second section we give brief overview of the related work. Then we present our prediction model that uses all four types of features and compensates the absence of a social graph by constructing a hidden graph of influence. Finally, experimental results are reported.

## *2. Related work*

Area of information propagation research contains a vast amount of problem formulations. J. Cheng et al. [2] predict for the spreading information cascade whether its size will double. Y. Yang et al. [3] offer a social model, RAIN, to predict the social roles of users that influence information diffusion. J. Yang and J. Leskovec [4] model the influence of the node on the rate of diffusion. For a more detailed survey on the topic, refer to [5].

We focus on two aspects of information diffusion: propagation modelling and prediction tasks.

Propagation modelling assumes that there is hidden graph of influence between network nodes. Vertices are sources of information, while the edge ($u$, $v$) means that the source $u$ affects $v$. When a social network is explicitly present we consider that information is distributed according to some algorithm in this graph. Threshold and cascade models are the most popular models of information propagation. They differ in the rules of information transfer between the nodes.

In the threshold model, edges of the graph have weights, and each user $u$ is given a threshold value $\phi_u$. User u gets the information if the sum of its active neighbours becomes equal to or greater than $\phi_u$. The cascade model makes two assumptions. First, the pairwise influences of the nodes on each other are independent, and second, any user $u$ has only one chance to transmit information to its neighbour $v$, regardless of the result in the next steps, the node $u$ will not affect $v$. At each new iteration $i$, every node $u$ that received information on the previous iteration ($i$-1) is trying to send it to all its neighbours $v$ that do not have this information yet with a specified probability $p_{uv}$. In both models the process ends when the number of knowledgeable users does not change.

Several models were proposed for constructing an influence graph in the absence of a social graph [6-8]. Most known of them is the paper by Gomez, Leskovec and Krause [6], where authors proposed efficient algorithm for building the graph of influence called NetInf. The algorithm build a graph $\hat{G}$ that maximises probability of observed cascades:

$$\hat{G} =_{|C|\leq m} P(C|G),$$

where the maximisation is over all directed graphs $G$ of at most $m$ edges.

In case of prediction tasks, methods based on extracting features [9, 10] and deep learning methods [11-13] are used to predict how information will spread in the network based on the previous propagations. These methods do not necessarily need a social graph.

As far as we know. there is no model that combines solutions for these two directions to improve the popularity prediction. In this paper, we apply the feature-based method for solving information propagation problem and build a graph of influence if the network does not have a social graph.

### *3. Prediction model*

In this section, we describe the proposed model. The model consists of two parts. At the first stage it reconstructs a hypothesised graph of influence using NetInf algorithm based on a given set of cascades. Then, at the prediction stage, we extract four types of features from the data and apply a classifier to predict the popularity. We use XGBoost classifier for predicting where parameters *max_depth* and *min_child_weight* were tuned using cross-validation. Other parameters were taken by default. We will use two metrics for evaluation of the model: precision and recall.

Further we consider each step in more details.

## 3.1 Influence graph reconstruction

Recall the assumption that some hidden relationships between the users affect the information propagation. In this work, such relations have the form of a directed graph of influence. The nodes of the graph are information sources (such as news media or users), the edge from source $u$ to source $v$ means that there is an influence of $u$ on $v$. In several applications, an existing social graph can be used as the influence graph. For example, in Twitter or Sina Weibo, a graph edge is defined as a subscription of one user to another. However, in most information propagation cases, an explicit social graph is absent and therefore, the influence network is hidden. In this paper, we use NetInf [6] algorithm to construct such a graph of influence.

NetInf is an iterative greedy algorithm which maximises the likelihood function to find the most cinfluential edges in the network. For a set of cascades and a graph $G$ constructed on the involved $G$ nodes, the probability $P_C$ that these cascades would propagate in $G$, is computed. For that, the probabilities that a single cascade propagates in a subtree of $G$, and that the cascade propagates in $G$ are defined. Then the likelihood function $F_C$ equivalent to $P_C$ is introduced, and the problem is reduced to maximising the given function at each step. At the first step an empty graph, where nodes are users of the network, is selected. Each step NetInf adds an edge with the greatest contribution to the likelihood function $F_C$. After $m$ iterations we have a graph withmmost influential edges in the network. NetInf code is implemented in C++ and is publicly available [14].

## 3.2 Feature extraction

We extract four types of features from the data: temporal features, structural features, content features, and features of early adopters.

Temporal and structural features are collected in the same way as B. Shulman et al. [1]. Temporal features of a cascade are the most significant ones for information distribution, as was reported in the most of previous works. Temporal features are associated with the speed of propagation at the early stages. Many of such features are focused on the speed of obtaining information. The time between receiving information by the *k*-th early adopter and the publication in the first source is considered. Average time between information acceptance for the first half of early publications and average time for the second half of early publications are added in order to reflect propagation attenuation at the early stage.The distribution process could also be affected by the time of the first publication. At certain moments of day, news can become more popular. Therefore, we also take into account the day of the week and time of the day of the first publication.

As for the structural features, we use the reconstructed graph of influence instead of a social one.

The text of news item also have an impact on their spreading. The distribution process depends on the influence of some users on others. Usually a user writes the text similar to his influencer's. Therefore, one of the features of the news items is the similarity of texts written at the early stage. Jaccard coefficient of similarity was used.

Topic modelling is also used to study the content features. Texts of the first five publications for each information flow were used for training. Each text was preprocessed: we made all text characters lowercase, removed all stop words and non-Russian letters, then all the words were stemmed. After that, we ran 50 iterations of the LDA model [15] for 50 topics.

As a result, a vector obtained from the text of the first publication of the cascade was added to the feature vector.

## 3.3 Summary of selected features

**Features of early adopters:**

- average number of publications per day;
- early adopter id;
- percent of news written by the source in which it posted the news item at an early stage.

**Temporal:**

- time interval between the k-th and the first information adoption among the early adopters;
- average time between information adoption for the first half (rounded down) of early publications;
- average time between information adoption for the second half of earlypublications;
- day of the week of the early stage publications;

- time of the day of the first publication of news.

**Structural:**

- the number of edges of the k-th early adopter;
- the number of nodes reachable in one step from all early adopters;
- the number of edges in the subgraph of early adopters;
- the number of edges of the k-th early adopter in the subgraph of the early adopters;
- average distance between nodes in the subgraph of the early adopters.

**Content:**

- news topic (feature vector obtained via topic modelling);
- similarity of the text of the first and the k-th early source;
- news category.

## *4. Experimental evaluation*

### 4.1 Data

The Lastfm dataset [1] was taken for the experiments. Lastfm is a music website, which have a social graph where users can listen to the music and mark the songs they like. More than 212 000 cascades were obtained for 450 000 users from the beginning of the creation of the service until February 2014. For each song a cascade is built and has the form $c = ((u_1, t_1);\ (u_2, t_2)m, \dots, (u_m, t_m))$, where $u_i$ is the user, $t_i$ is the time when the user liked the song.

To analyse content features and build a graph of influence, we collected 68 000 cascades for 2500 news publications at Yandex news service from January 2016. Yandex news service automatically combines news related to one topic into stories. Cascades were collected from these stories using publication date.

### 4.2 Experiments

Time $T$ was taken equal 28 days for Lastfm, for Yandex $T = \infty$. The number of early adopters was taken equal to $k = 5$. For that reason, only items that have at least 5 adoptions were used in the prediction model.

NetInf is an iterative algorithm. At each iteration it adds an edge to the graph, which it considers to be the most influential. We vary the number of NetInf iterations to see how it affects the quality of the model. Results are shown on fig. 1 where *X*-axis corresponds to the number of NetInf iterations. One can see that the both precision and recall increase up to a certain point, then, starting from some values, it stabilises. We would recommend $m = 20\ 000$ edges for this dataset as a compromise. If increase the number of edges, the model quality grows insignificantly while the overall complexity grows dramatically.

*Табл. 1. Предсказание каскадов для Яндекс*
*Table 1. Yandex cascade prediction*

| **Types of features** | ***Precision*** | ***Recall*** |
|---|---|---|
| Temporal | 0:685 ± 0:001 | 0:578 ± 0:006 |
| Temporal + Structural | 0:705 ± 0:002 | 0:640 ± 0:009 |
| Temporal + Structural + Early Adopters | 0:736 ± 0:002 | 0:675 ± 0:011 |
| Temporal + Structural + Early Adopters + Content | **0:750** ± 0:004 | **0:722** ± 0:008 |

We experimented with different types of features for our training model on Yandex dataset: temporal features, structural features, content ones and features of early adopters. Table 1 shows the results. At first, we used only temporal features, which were the most efficient in most of literature. When we add structural features from the graph built by NetInf, the precision and

recall of the model increased by 2% and 6%, respectively. Finally, when our model is given all four types of features described in section 3.3, precision and recall improve by 7% and 15%, respectively.

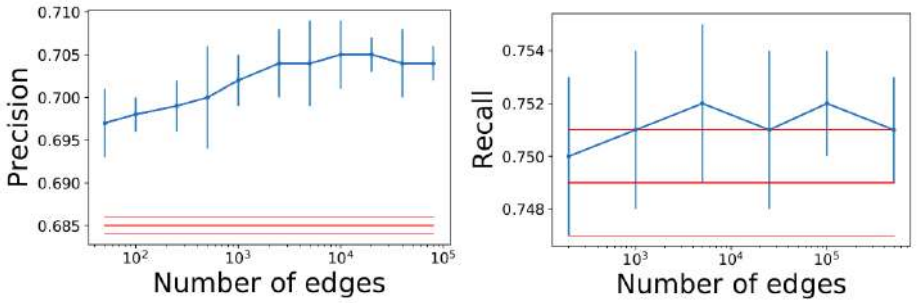


*Fig. 1. Precision and recall for the predicting model using temporal and structural features extracted from the hidden graph of influence with different number of edges using Yandex cascades. Red corresponds to results of the model that use only temporal features. Blue corresponds to results of the model that use temporal and structural features.*

We also checked how useful are the structural features based on the graph built by NetInf. For this purpose, we run our model on Lastfm dataset ignoring its social graph. Since the size of Lastfm social graph is more than 400 000 nodes, which was too computationally expensive for NetInf, we reduced its size. Therefore, we selected only the most active users who listened to more than 500 songs, which result to about 4000 nodes. Then we run NetInf on all the cascades from the dataset, containing only these nodes. Finally, we extracted structural features from the graph built by NetInf. We compared the effect of the obtained structural features with structural features extracted from the original social graph. Results for $m = 5000$ edges are presented in Table 2. We see that NetInf based features gives a slightly larger improvement to the prediction quality compared with social graph based features, although the difference is not significant. This means that the most influential connections between sources give the most impact on the information spread. It is also more efficient to extract structural features from the small graph of influence rather then the large social graph.

*Табл. 2. Предсказание каскадов для Lastfm*
*Table 2. Lastfm cascade prediction*

| **Types of features** | ***Precision*** | ***Recall*** |
|---|---|---|
| Temporal | 0:776 ± 0:003 | 0:749 ± 0:002 |
| Temporal + Structural (Social graph) | 0:778 ± 0:003 | 0:751 ± 0:002 |
| Temporal + Structural (NetInf) | 0:781 ± 0:003 | 0:752 ± 0:003 |

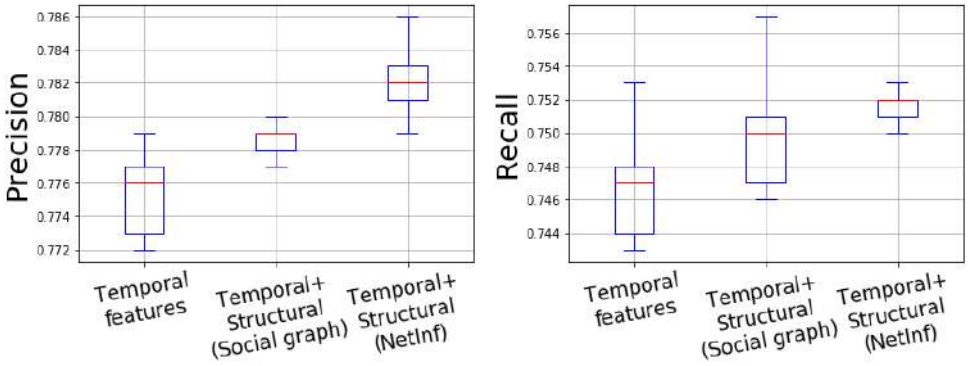


*Fig. 2. Precision and recall for the predicting model using temporal, temporal+structural (social graph) and temporal+structural (NetInf) features with 100 000 edges in the hidden graph using Lastfm cascades*

As with Yandex, we vary the number of NetInf iterations and observed a similar behaviour, see fig. 2. The increase of the number of iterations does not improve the prediction quality after about 5 000 edges and even starts to decrease after 100 000 edges.

### *5. Conclusion*

We proposed a model, which predicts at the early stage whether a news story will become larger than 50% of the rest stories in the given stream of news. If the network where the news propagate, is unavailable or does not exist, the model reconstructs a graph of influence and uses it to improve the prediction quality.

We evaluated our model on the Yandex news and Lastfm datasets. Yandex news has no social graph, while the Lastfm dataset has an underlying social network. Our main results are the following.

Structural features based on a constructed graph improves the prediction precision and recall by 2% and 6%, respectively. This confirms the assumption of existence of a hidden graph of influence.

Using the NetInf algorithm allowed to achieve similar or even better prediction quality than using the original social graph. This means that instead of observing a large social graph, it is better to take some small graph containing the most influential edges that will not worsen the prediction.

Using of all four types of features (temporal, structural, early adopters, and content) significantly improves the model compared to the use of only temporal features. Precision and recall improve by 7% and 15%, respectively.

## References / Список литературы

[1]. B. Shulman, A. Sharma, and D. Cosley. Predictability of popularity: gaps between prediction and understanding. In Proc. of the Tenth International AAAI Conference on Web and Social Media, 2016, pages 348–357.

[2]. J. Cheng, L. Adamic, P. A. Dow, J. M. Kleinberg, and J. Leskovec. Can cascades be predicted? In Proc. of the 23rd international conference on World Wide Web, 2014, pp. 925–936.

[3]. Y. Yang, J. Tang, C. W.-k. Leung, Y. Sun, Q. Chen, J. Li, and Q. Yang. Rain: social role-aware information diffusion. In Proc. of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015, pp. 367–373.

[4]. J. Yang and J. Leskovec. Modeling information diffusion in implicit networks. In Proc. of the 2010 IEEE 10th International Conference on Data Mining (ICDM, 2010), pp. 599–608.

[5]. Avetisyan A.A., Drobyshevskiy M.D., Turdakov D.Yu. Methods for Information Spread Analysis. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 6, 2018, pp. 199-220 (in Russian). DOI: 10.15514/ISPRAS-2018-30(6)-11 / Аветисян А.А., Дробышевский М.Д., Турдаков Д.Ю. Методы анализа информационных потоков в сети Интернет. Труды ИСП РАН, том 30, вып. 6, 2018 г., стр. 199-220.

[6]. M. Gomez Rodriguez, J. Leskovec, and A. Krause. Inferring networks of diffusion and influence. In Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2010, pp. 1019–1028.

[7]. M. G. Rodriguez, D. Balduzzi, and B. Schölkopf. Uncovering the temporal dynamics of diffusion networks. arXiv:1105.0697, 2011.

[8]. M. Gomez Rodriguez, J. Leskovec, and B. Schölkopf. Structure and dynamics of information pathways in online media. In Proceedings of the sixth ACM International Conference on Web Search and Data Mining, 2013, pp 23–32.

[9]. M. Jenders, G. Kasneci, and F. Naumann. Analyzing and predicting viral tweets. In Proceedings of the 22nd International Conference on World Wide Web, 2013, pp. 657–664.

[10]. S. Petrovic, M. Osborne, and V. Lavrenko. Rt to win! predicting message propagation in twitter. In Proc. of the Fifth International AAAI Conference on Weblogs and Social Media, 2011, pp. 586-589.

[11]. Q. Cao, H. Shen, K. Cen, W. Ouyang, and X. Cheng. Deephawkes: bridging the gap between prediction and understanding of information cascades. In Proc. of the 2017 ACM on Conference on Information and Knowledge Management, 2017, pp. 1149–1158.

[12]. C. Li, J. Ma, X. Guo, and Q. Mei. Deepcas: an end-to-end predictor of information cascades. In Proc/ of the 26th international conference on World Wide Web, 2017, pp. 577–586.
[13]. Q. Zhang, Y. Gong, J. Wu, H. Huang, and X. Huang. Retweet prediction with attention-based deep neural network. In Proc. of the 25th ACM International Conference on Information and Knowledge Management, 2016, pp. 75–84.
[14]. J. Leskovec. NETINF. Available at: http://snap.stanford.edu/netinf/, accessed 10,11.2019.
[15]. D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. Journal of Machine Learning Research, No. 3, 2003, pp. 993–1022.

## Информация об авторах / Information about authors

Арам Арутюнович АВЕТИСЯН, студент магистратуры факультета ВМК МГУ. Научные интересы: сбор данных, анализ информационных потоков в сети Интернет.

Aram Arutyunovich AVETISYAN, graduate student of the faculty of VMK at Moscow State University. Research interests: data collection, analysis of information flows on the Internet..

Михаил Дмитриевич ДРОБЫШЕВСКИЙ, младший научный сотрудник отдела информационных систем. Научные интересы: модели случайных графов, генерация сложных сетей с сохранением графовых свойств, машинное обучение.

Mikhail Dmitrievich DROBYSHEVSKY, Junior Researcher, Information Systems Department. Research interests: random graph models, generation of complex networks with preservation of graph properties, machine learning.

Денис Юрьевич ТУРДАКОВ, кандидат физико-математических наук, заведующий отделом информационных систем ИСП РАН, доцент кафедры системного программирования МГУ. Научные интересы: обработка естественного языка, машинное обучение, интеллектуальный анализ данных, анализ социальных сетей, распределенная обработка данных.

Denis Yuryevich TURDAKOV, Ph.D. in Physics and Mathematics, Head of the Information Systems Department at ISP RAS, Associate Professor of the System Programming Department of Moscow State University. Research interests: natural language processing, machine learning, data mining, social network analysis, distributed data processing.

DOI: 10.15514/ISPRAS-2019-31(5)-11

# Проактивная разметка примеров для адаптации к домену

[1] *М. А. Рындин, ORCID: 0000-0002-7504-3975 <mxrynd@ispras.ru>*
[1,2] *Д. Ю. Турдаков, ORCID: 0000-0001-8745-0984 <turdakov@ispras.ru>*
[1] *Институт системного программирования им. В.П. Иванникова 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*

**Аннотация.** В статье приводятся исследование возможности переноса знаний в целевой домен из другого, но близкого домена-источника с помощью проактивного обучения. Исследуется применимость использования модели машинного обучения, обученной на домене-источнике, как бесплатного ненадежного оракула для определения сложности примера из целевого домена и принятии решения о необходимости его разметки надежным экспертом. Представлен алгоритм такой разметки, одной из особенностей этого алгоритма является его возможность работы с любым классификатором, имеющим вероятностную интерпретацию выхода. Экспериментальное тестирование на наборе данных из отзывов на продукты Амазон подтверждает эффективность предложенного метода.

**Ключевые слова:** адаптация к домену; проактивное обучение

**Для цитирования:** Рындин М.А., Турдаков Д.Ю. Проактивная разметка примеров для адаптации к домену**.** Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 145-152. DOI: 10.15514/ISPRAS-2019-31(5)-11

## Domain adaptation by proactive labeling

[1] *M.A. Ryndin, ORCID: 0000-0002-7504-3975 <mxrynd@ispras.ru>*
[1,2] *D.Y. Turdakov, ORCID: 0000-0001-8745-0984 <turdakov@ispras.ru>*
[1] *Ivannikov Institute for System Programming of RAS, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*
[2] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*

**Abstract.** Getting tagged data is an expensive and time-consuming process. There are several approaches to how to reduce the number of examples needed for training. For example, the methods used in active learning are aimed at choosing only the most difficult examples for marking. Using active learning allows to achieve results similar to supervised learning, using much less labeled data. However, such methods are often dispersive and highly dependent on the choice of the initial approximation, and the optimal strategies for choosing examples for marking up either depend on the type of classifier or are computationally complex. Another approach is domain adaptation. Most of the approaches in this area are unsupervised and are based on approximating the distribution of data in domains by solving the problem of optimal transfer

or extraction of domain-independent features. Supervised learning approaches are not resistant to changes in the distribution of the target variable. This is one of the reasons why the task of semis-supervised domain adaptation is posed: there are labeled data in the source domain, a lot of unlabeled data in the target domain and the ability to get labels for some of the data from the target domain. In this work, we show how proactive labeling can help transfer knowledge from one source domain to a different but relative target domain. We propose to use a machine learning model trained on source domain as a free fallible oracle. This oracle can determine complexity of a training example to make several decisions. First, this example should be added to training dataset. Second, do we have enough knowldge learnt from source to label this example ourself or we need to call a trusted expert? We present an algorithm that utilize this ideas and one of its features is ability to work with any classifier that has probabilistic interpretation of its outputs. Experimental evaluation on Amazon review dataset establish the effectiveness of proposed method.

**Keywords:** domain adaptation; proactive learning

**For citation:** Ryndin M.A., Turdakov D.Y. Domain adaptation by proactive labeling. Trudy ISP RAN/Proc. ISP RAS, vol.31, issue 5, 2019, pp. 145-152 (in Russian). DOI: 10.15514/ISPRAS-2019-31(5)-11

## *1. Введение*

Получение размеченных данных – дорогостоящий и трудозатратный процесс. Существует несколько подходов к тому, как снизить количество примеров, необходимых для обучения.

Например, приёмы, использующиеся в активном обучении, направлены на выбор для разметки только наиболее трудных примеров. Использование активного обучения позволяет добиться результатов, аналогичных обучению с учителем, используя намного меньше размеченных данных. Однако, часто такие методы дисперсионны и сильно зависят от выбора начального приближения, а оптимальные стратегии выбора примеров для разметки либо зависят от вида классификатора [1], либо вычислительно сложны [2]. Еще один открытой проблемой является активная разметка примеров несколькими слабоквалифицированными аннотаторами – объединение идей активного обучения и краудсорсинга [3].

Другим подходом является адаптация к домену (предметной области). Задача обычно формулируется следующим образом: даны размеченные данные для задачи в одной предметной области, которая похожа на целевую и требуется построить решение для целевой предметной области, используя эти данные. Большая часть подходов в этой области являются алгоритмами обучения без учителя и основаны на сближении распределения данных в доменах с помощью решения задачи оптимального переноса [4] или извлечения доменно-независимых признаков с помощью, например, понижения размерности [5] или состязательного обучения [6].

Стоит отметить, что природа изменений в распределениях источника и целевого домена может варьироваться: может меняться только распределение примеров, может меняться распределение целевой переменной, эти типы изменений могут происходить одновременно. Подходы, использующие обучение без учителя, плохо справляются с

изменениями, затрагивающими распределение целевой переменной. Это является одной из причин, по которой ставится задача адаптации с учителем: имеются размеченные данные в домене-источнике, много неразмеченных данных в целевом домене и возможность получить метки для части данных из целевого домена. Для решения этой задачи естественным выглядит объединение методов активного обучения и адаптации к домену [7]. Одной из проблем приведённых исследований является зависимость алгоритма от типа модели машинного обучения. Авторы исследуют линейные модели, в то время как использование нелинейных моделей обычно позволяет добиться лучших результатов.

Обычно активное обучение предполагает идеализированную модель среды: имеется один всегда доступный и никогда не ошибающийся оракул, всё исследование сфокусировано на выборе примеров, которые дать ему на разметку. Проактивное обучение [8] снимает эти ограничения, исследователи рассматривают проблему выбора примеров для разметки при разных моделях ошибки оракула, при наличии нескольких оракулов с, возможно, разной стоимостью разметки. В работе [9] исследуется идея проактивной разметки примеров несколькими различными оракулами, авторы показывают, как работать с признаками, отсутствующими в домене-источнике. Однако их решение существенно использует линейность модели машинного обучения, что сказывается на итоговом качестве.

Также большинство алгоритмов активного обучения предполагают многочисленное обучение модели на вновь выбранных для разметки данных. Такой подход может привести к большим временным затратам при обучении сложных нелинейных моделей, поэтому для реального использования необходим алгоритм, минимизирующий количество циклов выбора примеров.

В данном исследовании предлагается использовать модель, построенную на домене-источнике, как дополнительный оракул. Предложенный алгоритм способен работать с произвольными (не только линейными) моделями, имеющими вероятностную интерпретацию выходов. Предложенный алгоритм позволяет добиться показателей качества, близких к качеству внутри домена, выбрав на разметку лишь небольшую часть примеров.

В следующем разделе будет приведена общая схема решения. Затем будет зафиксирована модель оракулов и ограничен класс моделей, с которыми способен работать алгоритм. В четвертом разделе приведено полное описание шагов алгоритма. Пятый раздел посвящен результатам экспериментального тестирования алгоритма.

## *2. Общая схема предлагаемого решения*

На рис. 1 представлена общая схема предложенного подхода, который состоит из 3 шагов.

1. Обучение модели на домене-источнике. Полученную модель обозначим $\boldsymbol{M_s}$.
2. Проактивная разметка с помощью обученной модели $\boldsymbol{M_s}$ и надежного оракула (эксперта или группа экспертов в предметной области) части примеров из целевого домена.
3. Обучение целевой модели на полученном наборе данных.

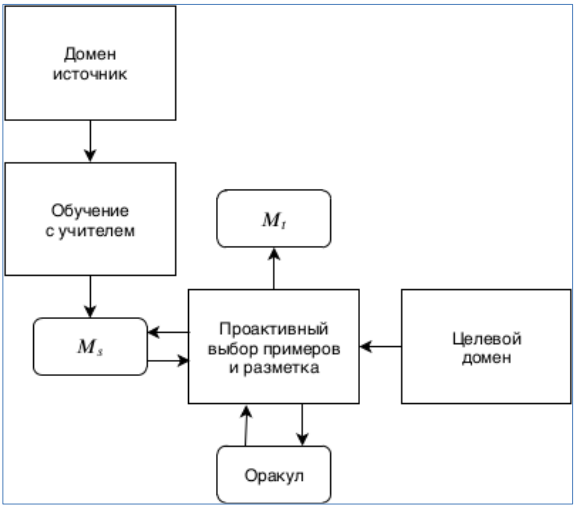


*Рис. 1. Диаграмма предложенного метода*
*Fig. 1. Block diagram of proposed algorithm*

*Табл. 1. Обозначения*
*Table 1. Definitions*

| | |
|---|---|
| $X_s$ | Примеры из домена-источника |
| $y_s$ | Целевая переменная для примеров из домена-источника |
| $X_t$ | Примеры из целевого домена |
| $M_s$ | Модель, построенная на данных из домена-источника |
| $M_t$ | Целевая модель |
| $O$ | Платный оракул |
| $p^j = M_s(x^j)$ | Вероятность принадлежности к классу «1» для данного примера $x^j \in X_t$, предсказанная бесплатным оракулом |
| $\tilde{y}^j = 1(p^j > 0.5)$ | Класс для данного примера $x^j \in X_t$, предсказанный бесплатным оракулом |
| $C = 0.5 - p^j \vee$ | Уверенность бесплатного оракула в данном примере $x^j \in X_t$ |
| $\theta$ | Гиперпараметр, граница уверенности бесплатного оракула, до которой мы ему верим |

### *3. Модели оракулов*

Используется два оракула: надежный, который можно рассматривать как группу экспертов (назовем его $O$) и ненадежный, который является моделью машинного обучения, обученной на домене-источнике (назовем его $M_s$). Опишем их модель.

- $\boldsymbol{O}$ – платный, считаем, что стоимость разметки одного примера фиксирована и не зависит от сложности примера. $\boldsymbol{M_s}$ по своей природе бесплатный.
- $\boldsymbol{O}$ – надежный и не ошибается. $\boldsymbol{M_s}$ может ошибаться.

Для работы с $M_s$ надо ввести модель ошибки, которая зависит от примера. Выход многих моделей машинного обучения имеет вероятностную интерпретацию (к примеру, это верно для логистической регрессии, нейросетей с активацией в виде софтмакс или сигмоиды в последнем слое). Например, выход бинарного классификатора с активацией-сигмоидой – число в диапазоне от 0 до 1, равное вероятности принадлежности классу «1».

В этом исследовании зафиксируем использование моделей с таким выходом. Для них модель ошибки вводится естественным образом. Например, для бинарной классификации обычно считают, что пример принадлежит классу «1», если выход классификатора больше 0.5, иначе классу «0». Соответственно, чем ближе к 0.5 выход классификатора, тем менее он уверен в примере. Следовательно, величину $0.5 - |0.5 - p^j \vee |$, где $p^j$ – предсказанная вероятность, логично использовать как вероятность ошибки.

### *4. Алгоритм проактивного выбора примеров и разметки*

В табл. 1 представлены обозначения, которые будут использованы далее. Алгоритм 1 описывает шаги при проактивной разметке. В начале принимается решение о необходимости добавить данный пример в обучающую выборку. Для этого смотрим на уверенность бесплатного оракула в данном примере. Чем она меньше, тем более непохож данный пример на примеры из источника, значит этот пример характерен для целевого домена и должен попасть в обучающую выборку. Поэтому пример попадает на разметку (бесплатную или платную решится позже), если реализация случайной величины из распределения Бернулли с параметром $C$ оказалась равна 0. Это почти всегда происходит для сложных примеров, а около половины легких – отсеивается.

Этот шаг вводится сразу для нескольких целей.

- Данный подход похож на выбор примеров по уверенности классификатора из активного обучения. Однако здесь используется не уверенность самого классификатора, а её аппроксимация уверенностью классификатора над доменом-источником.
- Такой способ выбора примера не предполагает многочисленного обучения целевой модели, которое происходит в алгоритмах активного обучения.

- Отсеивание части похожих примеров является механизмом, позволяющим забывать особенности одного домена и лучше обучаться к особенности другого.

**Input:** $M_s$

**Initialization:** $\hat{X} = \{\}, \hat{y} = \{\}$

**for all** $x^j \in X_t$ **do**
  Считаем $p^j, C$
  **if** $Bernoulli(C) = 0$ **then**
    Считаем $\tilde{y}^j$
    $\hat{X} = \hat{X} \cup x^j$
    **if** $C \leq \theta$ **then**
      $\hat{y} = \hat{y} \cup O(x^j)$
    **else**
      $\hat{y} = \hat{y} \cup \tilde{y}^j$
    **end if**
  **end if**
**end for**

*Алгоритм 1. Предлагаемый алгоритм проактивной разметки*
*Algorithm 1. Proposed algorithm of proactive labeling*

После этого принимается решение о способе разметки – если уверенность в данном примере выше порога, то пример размечается с помощью $M_s$, иначе отдается $O$.

Выходом алгоритма является множество $\hat{X} \subset X_t$ и меток к примерам из этого множества $\hat{y}$. С помощью этих данных строится целевая модель $M_t$.

## *5. Эксперименты*

### 5.1 Набор данных

Эксперименты проводились на наборе данных «Amazon review dataset»[1], содержащем отзывы на различные товары на английском языке. Изначальная целевая переменная – оценка товара по пятибалльной шкале, была заменена на бинарную: оценки до 3 включительно – класс «0» (отрицательный отзыв), более 3 – класс «1» (положительный отзыв).

Сравнение производилось со статьей [9], значения достоверности взяты из неё.

### 5.2 Используемая модель машинного обучения

При построении классификатора использовалась идея переноса знаний [10] с помощью неглубокой рекуррентной нейронной сети (2 скрытых LSTM слоя и выходной слой с активацией-сигмоидой) над предобученной[2] на Wikipedia языковой моделью fasttext [11].

[1] http://jmcauley.ucsd.edu/data/amazon/
[2] https://fasttext.cc/docs/en/english-vectors.html

Выделим источники случайности всего решения:

- Модели $\boldsymbol{M_s}$ и $\boldsymbol{M_t}$ как нейросети зависят от начальной инициализации весов.
- Выбор примеров в обучающую выборку случаен – серия испытаний Бернулли.
- $\boldsymbol{X_t}$ случайным образом перемешивается и разбивается на выборку для обучения (кандидаты на разметку) и теста в соотношении 9 к 1.

При каждом запуске алгоритма начальные значения генератора псевдослучайных чисел для каждого из источников изменялись. Представленные далее результаты являются осреднением по 15 запускам и округлены до первого значимого знака.

*Табл. 2. Результаты на «Amazon review»*
*Table 2. Results on «Amazon review»*

| Источник | Целевой домен | Достоверность, предложенный метод | Достоверность, без адаптации | Достоверность, достигаемая внутри домена | Достоверность, [9] |
|---|---|---|---|---|---|
| *B* | *E* | 90.7 ± 0.2 | 88.6 ± 0.1 | 91.2 ± 0.2 | 78.4 |
| | *K* | 89.9 ± 0.4 | 86.5 ± 0.2 | 91.3 ± 0.2 | 78.6 |
| *E* | *E* | 90.1 ± 0.2 | 87.0 ± 0.1 | 90.9 ± 0.1 | 77.8 |
| | *K* | 90.2 ± 0.3 | 89.1 ± 0.1 | 91.3 ± 0. | 86.0 |
| *K* | *E* | 91.1 ± 0.4 | 88.9 ± 0.3 | 91.2 ± 0.2 | 70.1 |
| | *B* | 89.3 ± 0.2 | 84.8 ± 0.1 | 90.9 ± 0.1 | 73.2 |

### 5.3 Результаты

В качестве метрики использовалась достоверность (accuracy). Гиперпараметр $\theta$ равнялся 0.25.

Обозначения доменов: B–books, E–electronics, K–kitchen. Результаты представлены в табл. 2. В среднем после первичного отбора на разметку попадало около 55 ± 5%всех данных, при этом среднее число бесплатно размеченных примеров примерно в 4 раза больше числа обращений к оракулу. При этом средняя доля ошибочно размеченных примеров не превышает 5 ± 2 % от числа размеченных примеров.

Из табл. 2 видно, что предложенный алгоритм позволяет добиться результатов, близких к качеству, которого можно добиться, имея все метки для примеров из целевого домена.

## *6. Заключение*

В данной работе представлен алгоритм адаптации к домену с учителем, использующий проактивную разметку. Этот алгоритм способен работать с любыми моделями машинного обучения, имеющими вероятностную интерпретацию выхода. Экспериментально показано, что алгоритм способен строить модели, близкие к качеству внутри домена, выбирая лишь часть примеров для разметки.

## Список литературы / References

[1] Cai Wenbin, Zhang Yexun, Zhang Ya, Zhou Siyuan, Wang Wenquan, Chen Zhuoxiang, Ding Chris. Active Learning for Classification with Maximum Model Change, ACM Transactions on Information

Systems, vol. 36, issue 2, 2017, pp. 15:1–15:28.
[2] Ozan Sener, Silvio Savarese. Active Learning for Convolutional Neural Networks: A Core-Set Approach. arXiv:1708.00489, 2017.
[3] Гилязев Р.А., Турдаков Д.Ю. Активное обучение и краудсорсинг: обзор методов оптимизации разметки данных. Труды ИСП РАН, том 30, вып. 2, 2018 г, стр. 215-250 / Gilyazev R.A., Turdakov D.Y. Active learning and crowdsourcing: a survey of annotation optimization methods. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 2, 2018, pp. 215-250 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-11.
[4] Nicolas Courty, Rémi Flamary, Devis Tuia, Alain Rakotomamonjy. Optimal Transport for Domain Adaptation. arXiv:1507.00504, 2015.
[5] Minmin Chen, Zhixiang Eddie Xu, Kilian Q. Weinberger, Fei Sha. Marginalized Denoising Autoencoders for Domain Adaptation. arXiv:1206.4683, 2012
[6] Yaroslav Ganin, Victor Lempitsky. Unsupervised Domain Adaptation by Backpropagation. Proceedings of the 32nd International Conference on Machine Learning, 1180–1189, 2015.
[7] Rai Piyush, Saha Avishek, Hal Daumé III, Venkatasubramanian Suresh. Domain Adaptation Meets Active Learning. Proceedings of the NAACL HLT 2010 Workshop on Active Learning for Natural Language Processing, 2010, 27–32.
[8] Pinar Donmez and Jaime G. Carbonell, From Active to Proactive Learning Methods. Advances in Machine Learning I. Springer, Berlin, Heidelberg, 2010. 97-120.
[9] Krishnapuram Raghu, Rajkumar Arun, Acharya Adithya, Dhara Nikhil, Goudar Manjunath, Sarashetti Akshay P. Online Domain Adaptation by Exploiting Labeled Features and Pro-active Learning. Proceedings of the ACM India Joint International Conference on Data Science and Management of Data, 2018.
[10] Howard Jeremy, Ruder Sebastian. Universal Language Model Fine-tuning for Text Classification. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2018, 328–339.
[11] Armand Joulin, Edouard Grave, Piotr Bojanowski, Tomas Mikolov. Bag of Tricks for Efficient Text Classification. CoRR, abs/1607.01759, 2016.

## Информация об авторах / Information about authors

Денис Юрьевич ТУРДАКОВ, кандидат физико-математических наук, заведующий отделом информационных систем ИСП РАН, доцент кафедры системного программирования МГУ. Научные интересы: обработка естественного языка, машинное обучение, интеллектуальный анализ данных, анализ социальных сетей, распределенная обработка данных.

Denis Yuryevich TURDAKOV, Ph.D. in Physics and Mathematics, Head of the Information Systems Department at ISP RAS, Associate Professor of the System Programming Department of Moscow State University. Research interests: natural language processing, machine learning, data mining, social network analysis, distributed data processing.

Максим Алексеевич РЫНДИН, аспирант ИСП РАН. Научные интересы: методы адаптации к домену и переноса знаний, онлайн обучение, обработка текстов на естественном языке, генеративные модели, активное и проактивное обучение, анализ социальных сетей.

Maxim Alekseevich RYNDIN, PhD student of ISP RAS. Research interests: methods for adapting to the domain and transferring knowledge, online learning, natural language processing, generative models, active and proactive learning, analysis of social networks.

**DOI: 10.15514/ISPRAS-2019-31(5)-12**



# Применение i-векторов для автоматизированного определения уровня близости языков

*А.У. Берзинь, ORCID: 0000-0002-3313-5935 <ansis@latnet.lv>*

*Латвийский университет,*
*LV-1003, Латвия, г. Рига, ул. Московская, д. 54*

**Аннотация**. В статье рассказывается о результатах применения i-векторных методов распознавания речи для задания расстояния между языками. В качестве входных данных используются фонограммы спонтанной речи. Эксперименты проводятся на звукозаписях латышских и латгальских говоров, но методы применимы и к любым другим идиомам.

**Ключевые слова:** речь; идиома; язык; диалект; i-вектор; и-вектор; фонограмма; близость языков; расстояние между языками

**Для цитирования:** Берзинь А.У. Применение i-векторов для автоматизированного определения уровня близости языков. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 153-164. DOI: 10.15514/ISPRAS-2019-31(5)-12

# Usage of i-Vectors for Automated Determination of a Similarity Level between Languages

*A.A. Bērziņš, ORCID: 0000-0002-3313-5935 <ansis@latnet.lv>*

*University of Latvia,*
*54, Moscow str., Riga, LV-1003, Latvia*

**Abstract.** The article describes results of applying i-vectors-based (both LID and SID) speech identification methods to define a kind of a distance between languages (in a wide sense of the word – including dialects and any other forms of spoken language). Spontaneous speech recordings of many enough speakers of languages are used on the input of the method. The experiments were carried out at recordings of Latvian and Latgalian dialects, but the method is applicable to any other idioms. Cosine similarity, Euclidean metric, standardized Euclidean metric, Jordan (or Chebyshov) metric and city block (or $L_1$) metric were tried out. Cosine similarity worked well for SID i-vectors, but for unknown reasons was senseless for LID i-vectors. Jordan metric worked well for LID, but was not good enough for SID i-vectors. Standardization of the Euclidean metric does not gave any improvement. Thus, the conclusions are: 1) both SID and LID vectors of full length recordings of spontaneous speech are characterizing and representing languages good enough to be used for detection of a distance between languages; 2) the best metrics for such tasks are Euclidean and $L_1$ (for arithmetic mean vectors computed from i-vectors of all informants coordinate by coordinate).

**Keywords:** speech; idiom; language; dialect; i-vector; LID; SID; recording; proximity of languages; distance between languages

**For citation:** Bērziņš A.A. Usage of i-Vectors for Automated Determination of a Similarity Level between Languages. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019, pp. 153-164 (in Russian). DOI: 10.15514/ISPRAS-2019-31(5)-12

## *1. Введение*

i-векторы – это относительно новый способ решения задач распознавания, который в настоящее время используется для распознавания объектов и других видов[1], но первоначально их ввели при поиске новых методов распознавания речи.

Первая относительно широко известная публикация, в которой озвучили данную идею (в контексте идентификации говорящего), вышла в 2009 году[2]. Название i-векторов в то время еще не появилось, в статье их пространство именуется пространством признаков полной изменчивости. В начале 2010 года сочетание «i-вектор» появилось как дополнительное название[3], но ко второй половине того же года им уже пользовались в полной мере, причём уже описывая именно задачи идентификации языка[4].

---

**1** Например, при распознавании символов: *We propose a novel text classification approach based on iVector, a newly developed concept in speaker verification. To a given text line, the iVector is a fixed-length feature vector representation, transformed from a high-dimensional supervector based on means of Gaussian mixture model (GMM), where the text dependent component is separated from a universal background model (UBM) and can be represented by a lowdimensional set of factors. We classify the text lines with a discriminative classifier - support vector machine (SVM) in iVector space. A baseline approach of text classification using GMM in feature space is also presented for evaluation purpose. Experimental results on an Arabic document database show accuracy of 92.04% for text line classification using the proposed method. Furthermore, the relative word error rate (WER) of 9.6% is decreased in optical character recognition (OCR) when coupled with the proposed iVector-SVM classifier. The proposed iVector-SVM approach is language independent, thus, can be applied to other scripts as well.* [2]

**2** *This paper presents a new speaker verification system architecture based on Joint Factor Analysis (JFA) as feature extractor. In this modeling, the JFA is used to define a new low-dimensional space named the total variability factor space, instead of both channel and speaker variability spaces for the classical JFA.* [3]

**3** *Based on this, we proposed a new speaker verification system based on factor analysis as a feature extractor. The factor analysis is used to define a new low-dimensional space named total variability space. In this new space, a given speech utterance is represented by a new vector named total factors (we also refer to this vector as "i-vector" in this paper).* [4]

**4** *..a new language identification system is presented based on the total variability approach previously developed in the field of speaker identification. Various techniques are employed to extract the most salient features in the lower dimensional i-vector space..* [5]

*..we described the application of the i-vector or total variability space approach to the language identification task. The i-vector representation is a data-driven approach for feature extraction that provides an elegant and general framework for audio classification and identification. It consists of mapping a sequence of frames for a given utterance into a low-dimensional vector space, referred to as the total variability space, based on a factor analysis technique.* [5]

Метод i-векторов основан на представлении моделей выражений гауссовой смеси со скрытой маломерной переменной и использовании изображения этого выражения в качестве вектора признаков в языковом классификаторе[5].

Бывают разные i-векторы в зависимости от того, какую лингвистическую информацию они содержат (например, акустическую, просодическую, фонотактическую), построены ли они на непрерывных или дискретных данных и предназначены ли для идентификации говорящего (SID), идентификации языка (LID) или других задач. Так что, на самом деле, мы могли бы даже говорить о целом ряде методов, но погружение в такие тонкости не является целью данной статьи.

## *2. Исходные данные*

В нашем распоряжении были собранные (записанные) нами звукозаписи спонтанной речи пяти идиомов (латвийских говоров) – один из Курляндии: Дундажской волости, и четыре из Латгалии: Аулеи, Бальтинова, Вилека и Рудзатов. Курляндия исторически была под немецким игом, поэтому местные говоры подверглись влиянию (нижне)немецкого языка, а северокурляндские говоры, в том числе и дундажский, содержут большой субстрат ливонского языка (принадлежащего к прибалтийско-финской подгруппе финно-угорских языков). Латгалия, в свою очередь, была под поляками, поэтому в латгальских говорах присутствует влияние польского языка, также – в силу близкого соседства и наличия белорусских и старообрядческих деревень – белорусского и русского. Бальтиновский и вилекский являются говорами северолатгальскими, которые от западнолатгальского рудзатского и южнолатгальского аулейского отличаются существенно – и морфологически, и лексически.

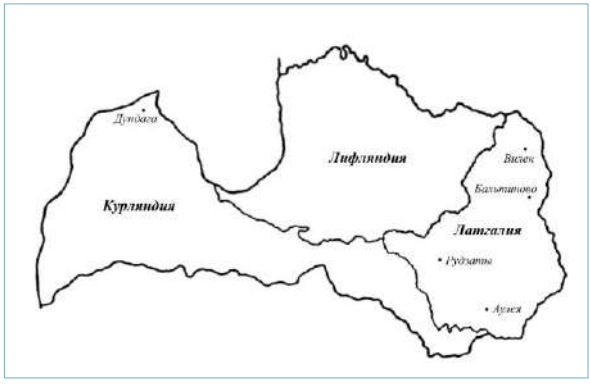


*Рис. 1. Расположение записанных говоров на карте Латвии*
*Fig. 1. The recorded dialects on the map of Latvia*

[5] *The most recent front-end subspace modeling technique known as iVector, which is a feature extraction model in the front-end of the language recognition system, has become the state-of-the-art technique in SID and was successfully adapted for the language recognition. The main idea of the iVector model in acoustic language recognition is to represent each utterance dependent GMM with a low-dimensional latent variable and use the low-dimensional representation of the utterance as a feature vector to the following language classifier.* [6]

Все звукозаписи собирались согласно заданным нами принципам сбора информации для автоматизированного анализа фонограмм [1], т.е., все записи были однородными, записанными однотипной аппаратурой (использовался динамический микрофон одностороннего направления, фиксированый на голове информанта), в условиях уменьшенного влияния внешних шумов. Все записи были мануально вычищены, удалению подверглись все посторонние звуки и голоса, оставив только прямую речь информанта. Качество записи – 44,1 кГц / 16 битов. В зависимости от требований конкретных скриптов по вычислению i-векторов, для фонограмм выполнялось понижение частоты дискретизации.

*Табл. 1. Характеристика набора фонограмм, используемого в эксперименте.*
*Table 1. Characteristics of recordings used in the experiment*

| *Говор* | *Минут* | *Информантов* | *Мужчин* | *Женщин* |
|---|---|---|---|---|
| **Аулея** | 95 | 14 | 8 | 6 |
| **Бальтиново** | 140 | 23 | 9 | 14 |
| **Дундага** | 161 | 17 | 4 | 13 |
| **Рудзаты** | 246 | 28 | 11 | 17 |
| **Вилек** | 238 | 30 | 11 | 19 |

Всех информантов просили рассказывать о быте, родителях, бабушках, дедушках, братьях, сёстрах, детях, других членах семьи, учёбе, работе, хозяйстве, службе в армии, свадьбах, праздниках, соседях и т. п. Т.е., в ходе сбора данных на традиционность и гомогенность лексики обращалось пристальное внимание.

Поэтому, учитывая однородность нашего многоговорного корпуса и в техническом, и в содержательном смысле, мы даже можем не постесняться его считать сопоставимым[6]. Доселе этот термин применялся только к текстовым корпусам, но мы считаем, что его можно применять и к речевым, и при таком применении наш корпус соответствует смыслу сопоставимости.

## *3. Эксперимент. Векторы SID*

Поскольку в нашем распоряжении были скрипты вычисления i-векторов, разработанные Брненским Техническим университетом (БТУ), мы, конечно, ими воспользовались. В 2015 году Речевая группа БТУ выступила с предложением создать общий стандарт голосовой биометрии – *Voice Biometry Standart* или VBS, поскольку различные используемые в настоящее время технические стандарты не позволяют быстро предоставлять данные и обмениваться ими. Предложение подкреплялось скриптами,

[6] *A comparable corpus is one which selects similar texts in more than one language or variety. There is as yet no agreement on the nature of the similarity, because there are very few examples of comparable corpora. ... The possibilities of a comparable corpus are to compare different languages or varieties in similar circumstances of communication, but avoiding the inevitable distortion introduced by the translations of a parallel corpus.* [12]

*A comparable corpus is a pair of corpora in two different languages, which come from the same domain.* [13]

написанными на питоне, для вычисления i-векторов в предложенном стандарте[7] (на вход можно подавать необработанные фонограммы речи, но при подаче заранее определённых интервалов голосовой активности – VAD или *Voice Activity Detection* – результаты лучше), потому что встроенный определитель активности очень примитивен. В [7] достаточно подробно описано и теоретическое обоснование VBS, и его техническая реализация, поэтому мы не будем это переписывать. Понятно, что биометрический стандарт предназначен для задач идентификации говорящего, то есть он фокусируется на особенности речи (в том числе голоса) определённого лица, т.е. i-векторы, генерируемые этим пакетом, называются SID (от *Speaker IDentification*) i-векторами. Теоретически они менее подходят для нашей задачи, но мы решили их испробовать, так как открытый стандарт и общедоступность ПО являются ключевыми факторами при выборе технологии. i-векторы мы рассчитывали для полных фонограмм спонтанной речи информантов данного языка, таким образом ожидая, что они, в силу большой продолжительности звукозаписи, будут характеризовать язык в целом, т.е., и фонетические, и морфологические, и лексические, и даже синтаксические особенности.

Косинусный коэффициент[8] (или мера Отиаи) был предложен в качестве наиболее перспективного метода оценки расстояния между i-векторами с самого начала – в изначальных публикациях об i-векторах[9]. Несмотря на постоянные поиски различных способов его улучшения[10], он всё равно пока остаётся основным методом оценки близости i-векторов. Поэтому для всех пар i-векторов говоров мы сначала вычислили косинусный коэффициент.

---

**7** *This standard is supposed to give formal description of the i-vector extraction algorithm. However, we provide a python demo package for i) better understanding of the properties and features of the extraction, and ii) for convenience, so that the user can immediately use the basic functions and do prompt customizations.* [7]

**8** *Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction.* [8]

**9** *We have proposed two new systems based on this new speech representation. The first system is an SVM-based system which uses the cosine kernel to compute the similarity between the total factors. The second system directly uses the value of the cosine distance computed between the target speaker factors and test total factors as a decision score. In this scoring, we removed the SVM from the decision process. One important characteristic of this approach is that there is no speaker enrolment, unlike in other approaches like SVM and JFA, which makes the decision process faster and less complex.* [4]

**10** *This paper deals with the problem of processing of I-vectors in the text-independent speaker verification systems. A new generalized cosine similarity optimization technique is proposed. The optimization is performed over sets of orthogonal and diagonal matrices.* [9]

*It is known that the equal-error-rate (EER) performance of a speaker verification system is determined by the overlap region of the decision scores of true and imposter trials. Also, the cosine similarity scores of the true or imposter trials produced by the state-of-the-art i-vector front-end approximate to a Gaussian distribution, and the overlap region of the two classes of trials depends mainly on their between-class distance. Motivated by the above facts, this paper presents a cosine similarity learning (CML) framework for speaker verification, which combines classical compensation techniques and the cosine similarity scoring for improving the EER performance. CML minimizes the overlap region by enlarging the between-class distance while introducing a regularization term to control the with-in class variance, which is initialized by a traditional channel compensation technique such as linear discriminant analysis.* [10]

*Табл. 2. Косинусный коэффициент между SID i-векторами фонограмм (значения округлены)*
*Table 2. Cosine similarity between SID i-vectors of our recordings (values rounded)*

| | Auleja | Baļtinova | Dundag | Rudzātys | Vileks |
|---|---|---|---|---|---|
| Auleja | 1 | 0,07 | -0,73 | -0,16 | -0,08 |
| Baļtinova | 0,07 | 1 | -0,63 | -0,33 | 0,68 |
| Dundag | -0,73 | -0,63 | 1 | 0,21 | -0,54 |
| Rudzātys | -0,16 | -0,33 | 0,21 | 1 | -0,45 |
| Vileks | -0,08 | 0,68 | -0,54 | -0,45 | 1 |

*Табл. 3. Углы в градусах от косинусного коэффициента между SID i-векторами наших фонограмм (значения округлены)*
*Table 3. Angles in degrees from cosine similarity on SID i-vectors of our recordings (values rounded)*

| | Auleja | Baļtinova | Dundag | Rudzātys | Vileks |
|---|---|---|---|---|---|
| Auleja | 0 | 86 | 137 | 99 | 94 |
| Baļtinova | 86 | 0 | 129 | 109 | 47 |
| Dundag | 137 | 129 | 0 | 78 | 122 |
| Rudzātys | 99 | 109 | 78 | 0 | 116 |
| Vileks | 95 | 47 | 122 | 116 | 0 |

Косинусный коэфициент по сути – величина косинуса, а арккосинус от него возвращает углы, которые также характеризуют вычисляемое расстояние. Поэтому визуально вообразить, что к чему ближе, а что от чего дальше, лучше всего можно, перейдя на углы в градусах, т.е., вычислив арккосинус от косинусного коэффициента. Представьте себе нулевую линию, проведенную на плоскости и на ней точку или центр; тогда угол, образованный лучами от этого центра и нулевого луча (правый луч нулевой линии), представляет соответствующее расстояние между двумя кривыми – чем меньше угол, тем языки ближе.

Если таким образом проанализировать 3-ю таблицу, то видно, что результаты являются осмысленными, т.е., между более близкими языками углы меньше, а более далёкими – больше.

Так бальтиновский и вилекский говоры оказываются ближайшей парой (47°). Расстояние между южно- и западнолатгальскими говорами – рудзатским и аулейским – меньше, чем между ними и севернолатгальскими. Дундага наиболее удалена от Аулеи, Бальтинова и Вилека. Единственная оценка, которая кажется в корне неверной, это расстояние между рудзатским и дундажским – оно, безусловно, не должно было быть меньше, чем расстояние между рудзатским и тремя остальными латгальскими говорами.

Интереса ради, дабы было с чем сравнить, мы для определения расстояния между i-векторами решили испробовать также евклидову и другие векторные метрики (например, жорданову[11]). Под влиянием [11] мы решили применить тоже метрику городского квартала. Так как все вышеупомянутые метрики заданы на векторном пространстве, то потребовалось привести наши многовекторные характеристики к одному вектору: мы это

[11] Она же – метрика Чебышёва, расстояние Чебышёва, равномерная метрика, sup-метрика, бокс-метрика.

сделали для каждого языка, покоординатно вычислив средний арифметический i-вектор из i-векторов информантов данного языка.

*Табл. 4. Евклидова метрика между SID i-векторами наших фонограмм (значения округлены)*
*Table 4. Euclidean metric between SID i-vectors of our recordings (values rounded)*

| | Auleja | Baļtinova | Dundag | Rudzātys | Vileks |
|---|---|---|---|---|---|
| Auleja | 0 | 8,04 | 11,42 | 8,71 | 9,00 |
| Baļtinova | 8,04 | 0 | 9,78 | 7,56 | 6,56 |
| Dundag | 11,42 | 9,78 | 0 | 9,27 | 9,13 |
| Rudzātys | 8,71 | 7,56 | 9,27 | 0 | 7,24 |
| Vileks | 9,00 | 6,56 | 9,13 | 7,24 | 0 |

*Табл. 5. Нормализованная евклидова метрика между SID i-векторами наших фонограмм (значения округлены)*
*Table 5. Standardized Euclidean metric between SID i-vectors of our recordings (values rounded)*

| | Auleja | Baļtinova | Dundag | Rudzātys | Vileks |
|---|---|---|---|---|---|
| Auleja | 0 | 0,480 | 0,663 | 0,515 | 0,542 |
| Baļtinova | 0,480 | 0 | 0,582 | 0,458 | 0,405 |
| Dundag | 0,663 | 0,582 | 0 | 0,547 | 0,549 |
| Rudzātys | 0,515 | 0,458 | 0,547 | 0 | 0,443 |
| Vileks | 0,542 | 0,505 | 0,549 | 0,443 | 0 |

*Табл. 6. Жорданова метрика между SID i-векторами наших фонограмм (значения округлены)*
*Table 6. Jordan metric between SID i-vectors of our recordings (values rounded)*

| | Auleja | Baļtinova | Dundag | Rudzātys | Vileks |
|---|---|---|---|---|---|
| Auleja | 0 | 1,20 | 1,39 | 1,24 | 1,36 |
| Baļtinova | 1,20 | 0 | 1,35 | 0,95 | 0,87 |
| Dundag | 1,39 | 1,35 | 0 | 1,10 | 1,04 |
| Rudzātys | 1,24 | 0,95 | 1,10 | 0 | 0,94 |
| Vileks | 1,36 | 0,87 | 1,04 | 0,94 | 0 |

*Табл. 7. Метрика городского квартала или $L_1$ между SID i-векторами наших фонограмм (значения округлены)*
*Table 7. City block or $L_1$ metric between SID i-vectors of our recordings (values rounded)*

| | Auleja | Baļtinova | Dundag | Rudzātys | Vileks |
|---|---|---|---|---|---|
| Auleja | 0 | 155 | 223 | 170 | 173 |
| Baļtinova | 155 | 0 | 188 | 149 | 127 |
| Dundag | 223 | 188 | 0 | 182 | 178 |
| Rudzātys | 170 | 149 | 182 | 0 | 142 |
| Vileks | 173 | 127 | 178 | 142 | 0 |

Из табл. 4–7 видно, что в нашем случае наихудшие результаты (хоть и не совсем плохие) показала жорданова метрика: Аулея и для Вилека, и для Рудзат оказалась гораздо дальше Дундаги.

Метрики $L_1$ и евклидова (как нормализованная, так и обыкновенная, поскольку нормализация на результаты существенно не повлияла) обе выглядят одинаково хорошо и – главное – даже лучше, чем косинусный коэффициент: Вилек и Бальтиново – самые близкие, Дундага – по отношению ко всем латгальским говорам – самая дальняя.

Единственный вопрос, который возникает: почему Аулея к Бальтинову оказывается ближе чем к Рудзатам? Это может быть ошибкой метрики, неадекватностью данных, но также и объективной оценкой, которая учитывает некоторые диалектальные нюансы, которые в теоретических сравнениях обычно игнорируются. Чтобы ответить на этот вопрос, необходимы дополнительные эксперименты с бóльшим количеством данных и бóльшим количеством говоров.

## *4. Эксперимент. Векторы LID*

Другой вид i-векторов, предназначенный для идентификации языка, называется LID (от *Language IDentification*).

Сперва мы провели предварительный эксперимент, дабы убедиться, что i-векторный метод распознавания языка эффективен для наших диалектальных звукозаписей: во время стажировки в Брненском Техническом университете мы дали Олдриху Плоту, научному сотруднику Исследовательской группы по обработке речи наши фонограммы (он их попросил для своих экспериментов) и попросили заодно провести и опыты, интересующие нас.

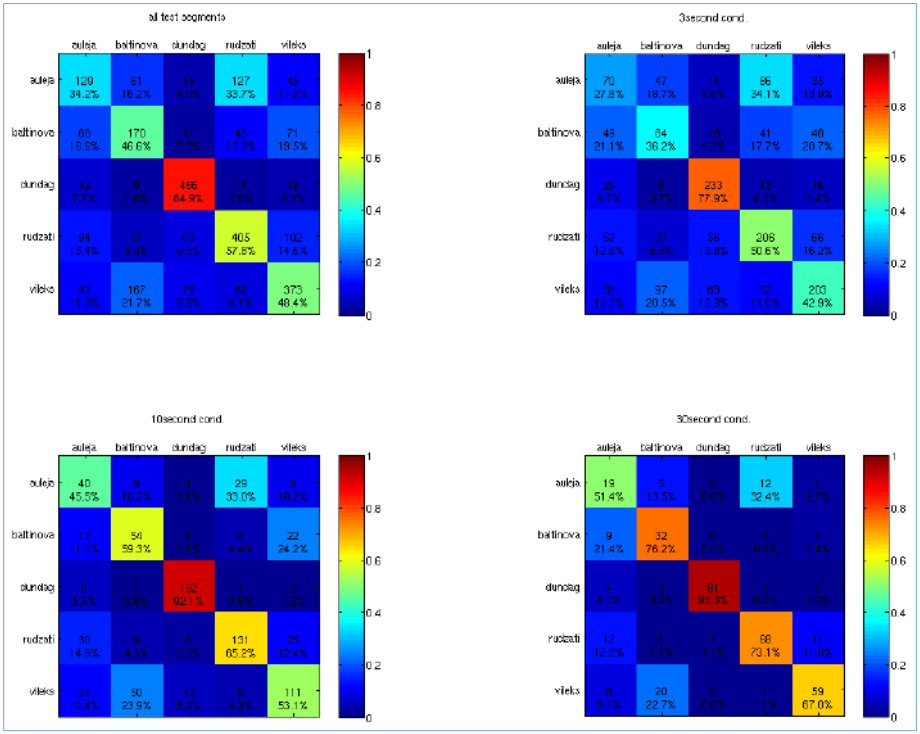


*Рис. 2. Результаты классификации наших звукоданных, предоставленые Исследовательской группой по обработке речи Брненского Технического университета*

*Fig. 2. Classification results of our audio data provided by the Speech Processing Research Group of the Technical University of Brno.*

Перед проведением эксперимента данные каждого говора были путём случайной выборки разделены на две части: бóльшую, обучающую часть и меньшую, проверочную часть. Затем i-векторы были рассчитаны для каждой части отдельно. После этого гауссовский линейный классификатор[12] обучался на i-векторах обучающей части, а на i-векторах проверочной части он в свою очередь применялся.

На рис. 2 для речевых сегментов разной длительности показано процентное распределение того, сколько проверочных данных было распределено правильно (т.е., правильно определён говор фонограммы) и сколько – неправильно. Как видим, результаты весьма близки к реальности: Дундага как наиболее отличающаяся определяется лучше всего; Рудзаты тоже отражены достоверно, то, что они «отдают» часть другим латгальским говорам, вполне объяснимо; Бальтиново и Вилек, учитывая их близость, также показывают относительно хорошие результаты, причём большая часть разницы «отдаётся» друг другу – между собой; единственное, что удивляет, это сравнительно плохие результаты Аулеи «в пользу» Рудзатов.

Учитывая хорошие результаты брненцев, мы решили на их LID i-векторах, которые они нам любезно предоставили, провести свои эксперименты, которые мы до того провели на SID i-векторах. Мы ожидали, что результаты будут подобны SID i-векторным, но всё-таки чуть лучше, потому что LID i-векторы предназначены для решения задачи, более схожей с нашей.

*Табл. 8. Косинусный коэффициент между LID i-векторами фонограмм (значения округлены)*
*Table 8. Cosine similarity between LID i-vectors of our recordings (values rounded)*

| | Auleja | Baļtinova | Dundag | Rudzātys | Vileks |
|---|---|---|---|---|---|
| Auleja | 1 | 0,49 | -0,06 | 0,57 | -0,85 |
| Baļtinova | 0,49 | 1 | -0,24 | -0,05 | -0,49 |
| Dundag | -0,06 | -0,24 | 1 | 0,06 | -0,04 |
| Rudzātys | 0,57 | -0,05 | 0,06 | 1 | -0,82 |
| Vileks | -0,85 | -0,49 | -0,04 | -0,82 | 1 |

*Табл. 9. Углы в градусах от косинусного коэффициента между LID i-векторами наших фонограмм (значения округлены)*
*Table 9. Angles in degrees from cosine similarity on LID i-vectors of our recordings (values rounded).*

| | Auleja | Baļtinova | Dundag | Rudzātys | Vileks |
|---|---|---|---|---|---|
| Auleja | 0 | 61 | 93 | 55 | 148 |
| Baļtinova | 61 | 0 | 104 | 93 | 120 |
| Dundag | 93 | 104 | 0 | 87 | 92 |

[12] *The next step in an LID system is training of language models using the generated iVectors. .. ..it is enough to have class (language) likelihoods so that we can make an optimal Bayesian decision on the language of a trial. The optimal Bayesian decision could be made if our LID system delivers optimal likelihoods for the languages of interest. E.g. if the task is to minimize the probability of language misclassification, we can select the most likely language, where the language posteriors can be obtained using Bayes rule from the priors and likelihoods. Having iVectors, our back-end would be a single multi-class probabilistic classifier (e.g. multi-class logistic regression, Gaussian linear classifier and etc.) that takes iVectors as inputs and, by definition, delivers class likelihoods. ..* [6] Далее в [6] выводятся формулы гауссовского линейного классификатора.

| | | | | | |
|---|---|---|---|---|---|
| Rudzātys | 55 | 93 | 87 | 0 | 145 |
| Vileks | 148 | 120 | 92 | 145 | 0 |

К нашему великому удивлению, результаты косинусного коэффициента оказались совершенно бессмысленными. Установить причины этого нам пока не удалось.

Несмотря на неудачу, мы решили на LID i-векторах испробовать и остальные расстояния, применённые для SID i-векторов. Для евклидовой и $L_1$ метрик результаты были похожими на SID i-векторные. Интересно, однако, что жорданова метрика, которая для SID i-векторов была не слишком адекватной, вела себя намного лучше на LID i-векторах – без каких-либо наглядных проблем, как в случае с SID i-векторами, и, можно сказать, почти так же хорошо, как евклидова и $L^1$.

*Табл. 10. Евклидова метрика между LID i-векторами наших фонограмм (значения округлены)*
*Table 10. Euclidean metric between LID i-vectors of our recordings (values rounded)*

| | | Auleja | Baļtinova | Dundag | Rudzātys | Vileks |
|---|---|---|---|---|---|---|
| Auleja | | 0 | 6,07 | 9,59 | 6,22 | 6,76 |
| Baļtinova | | 6,07 | 0 | 8,75 | 6,91 | 4,71 |
| Dundag | | 9,59 | 8,75 | 0 | 8,39 | 8,45 |
| Rudzātys | | 6,22 | 5,91 | 8,39 | 0 | 6,14 |
| Vileks | | 6,76 | 4,71 | 8,45 | 6,14 | 0 |

*Табл. 11. Нормализованная евклидова метрика между LID i-векторами наших фонограмм (значения округлены)*
*Table 11. Standardized Euclidean metric between LID i-vectors of our recordings (values rounded)*

| | Auleja | Baļtinova | Dundag | Rudzātys | Vileks |
|---|---|---|---|---|---|
| Auleja | 0 | 0,381 | 0,586 | 0,390 | 0,428 |
| Baļtinova | 0,381 | 0 | 0,543 | 0,377 | 0,303 |
| Dundag | 0,586 | 0,543 | 0 | 0,519 | 0,528 |
| Rudzātys | 0,390 | 0,377 | 0,519 | 0 | 0,394 |
| Vileks | 0,428 | 0,303 | 0,528 | 0,394 | 0 |

*Табл. 12. Жорданова метрика между LID i-векторами наших фонограмм (значения округлены)*
*Table 12. Jordan metric between LID i-vectors of our recordings (values rounded)*

| | Auleja | Baļtinova | Dundag | Rudzātys | Vileks |
|---|---|---|---|---|---|
| Auleja | 0 | 0,75 | 1,13 | 0,79 | 0,87 |
| Baļtinova | 0,75 | 0 | 1,31 | 0,75 | 0,59 |
| Dundag | 1,13 | 1,31 | 0 | 1,11 | 1,06 |
| Rudzātys | 0,79 | 0,75 | 1,11 | 0 | 0,81 |
| Vileks | 0,87 | 0,59 | 1,06 | 0,81 | 0 |

*Табл. 13. Метрика городского квартала или L1 между LID i-векторами наших фонограмм (значения округлены)*

*Table 13. City block or L1 metric between LID i-vectors of our recordings (values rounded)*

| | Auleja | Baļtinova | Dundag | Rudzātys | Vileks |
|---|---|---|---|---|---|
| Auleja | 0 | 120 | 188 | 123 | 131 |
| Baļtinova | 120 | 0 | 171 | 115 | 91 |
| Dundag | 188 | 171 | 0 | 164 | 167 |
| Rudzātys | 123 | 115 | 164 | 0 | 119 |

| | | | | | |
|---|---|---|---|---|---|
| Vileks | 131 | 91 | 167 | 119 | 0 |

## *5. Выводы*

Вследствие проведённых экспериментов мы убедились, что i-векторы достаточно хорошо характеризуют языки и поэтому могут использоваться для количественной оценки языковых различий, причём пользоваться можно как SID, так и LID i-векторами. Кроме того, несмотря на традицию применения косинусного коэффициента, более надёжно воспользоваться евклидовой или $L_1$ метриками.

Метод i-векторов не единственный метод, разрабатываемый нами для решения задачи численной оценки близости языков, в том числе и по звукозаписям. Поэтому для нас актуальна задача сравнения этих методов, отбора лучших из них и даже создания «надметода», обьединяющего наши разработки. Конечно, никаких «золотых стандартов»[13] в этой области не существует, особенно в контексте латышских говоров. Мы уже начали работу по разработке и применению метода экспертных оценок, который подходил бы под наши данные и методы, в том числе и описанный в этой статье. Но это уже тема отдельной публикации...

## Список литературы / References

[1]. A.A. Bērziņš. The Principles of Collection of Information for Automated Analyse of Audio Recordings. Tbilisi, Meridiani, 2011, pp. 39–46 (in Georgian and Russian) / А.У. Берзинь. Принципы сбора информации для автоматизированного анализа фонограм. Тбилиси, Меридиани, 2011 / ბერზინი ა. ინფორმაციის მოპოვების პრინციპები ფონოგრამების ავტომატური ანალიზისთვის. ქართული ენა და თანამედროვე ტექნოლოგიები, თბილისი, მერიდიანი, 2011

[2]. Zha Sh., Peng X., Cao H., Zhuang X., Natarajan P., Natarajan P. Text Classification via iVector Based Feature Representation. In Proc. of the 11th IAPR International Workshop on Document Analysis Systems, 2014, pp. 151-155.

[3]. Dehak N., Dehak R., Kenny P., Brummer N., Ouellet P., Dumouchel P. Support vector machines versus fast scoring in the low-dimensional total variability space for speaker verification. In Proc. of the Interspeech Conference, 2009, pp. 1559-1562.

[4]. Dehak N., Kenny P.J., Dehak R., Dumouchel P., Ouellet P. Front-End Factor Analysis for Speaker Verification. IEEE Transactions on Audio, Speech, And Language Processing, vol. 19, no. 4, 2011, pp. 788-798.

[5]. Dehak N., Torres-Carrasquillo P.A., Reynolds D., Dehak R. Language Recognition via Ivectors and Dimensionality Reduction. In Proc. of the Interspeech Conference, 2011, pp. 857-860.

[6]. Soufifar M. Subspace Modeling of Discrete Features for Language Recognition. Doctoral theses, Trondheim, NTNU, 2014.

[7]. Glembek O., Burget L., Matejka P. Voice Biometry Standard, Draft. Brno: Speech@FIT, 2015.

[8]. Han J., Kamber M., Pei J. Data Mining: Concepts and Techniques. 3rd Edition. Morgan Kaufmann, 2012, 800 p.

[9]. Drgas Sz., Dąbrowski A. Generalized cosine similarity in I-vector based automatic speaker recognition systems. In Proc. of the International Conference on Signal Processing: Algorithms, Architectures, Arrangements, and Applications, 2013, pp. 73-77.

[13] Например, наподобие описанных в [14].

[10]. Bai Zh., Zhang X.-L., Chen J. Cosine Metric Learning for Speaker Verification in the i-Vector Space. In Proc. of the Interspeech Conference, 2018, pp. 1126-1130.
[11]. Ghosh S., Vijay Girish K.V., Sreenivas T.V. Relationship between Indian Languages Using Long Distance Bigram Language Models. In Proc of the 9'th International Conference on Natural Language Processing, 2011, pp. 104-113.
[12]. Preliminary recommendations on Corpus Typology. EAGLES – Expert Advisory Group on Language Engineering Standards Guidelines, 1996. Available at: http://www.ilc.cnr.it/EAGLES96/corpustyp/corpustyp.html, 05.11.2019.
[13]. Comparable Corpora. MT Research Survey Wiki. University of Edinburgh. Available at: http://www.statmt.org/survey/Topic/ComparableCorpora, 05.11.2019.
[14]. Similarity (State of the art). ACL Wiki for Computational Linguistics. The Association for Computational Linguistics. Available at: https://aclweb.org/aclwiki/Similarity (State_of_the_art), 06.11.2019.

## Информация об авторе / Information about the author

Анс-Атаол Улдович БЕРЗИНЬ – магистр математических наук, завершающий свой труд над диссертацией по компьютерной лингвистике. Сферы научных интересов: лингвометрия, распознавание речи, машинный перевод, малые языки, лексикография, терминология, фольклор, сравнительное языкознание, этномузыкология, права человека, конституционное право, функциональный анализ.

Ansis Ataols BĒRZIŅŠ – Master of Mathematics, completing his work on thesis on computational linguistics. Research interests: linguometry, speech recognition, machine translation, endangered and low-resourced languages, lexicography, terminology, folklore, comparative linguistics, ethnomusicology, human rights, constitutional law, functional analysis.

**DOI: 10.15514/ISPRAS-2019-31(5)-13**



# Методы и средства разработки автоматизированных информационных систем на основе онтологии «Управление качеством программно-технических комплексов»

*А.В.Самонов, ORCID: 0000-0002-0390-4481 <a.samonov@mail.ru>*

*Военно-космическая академия им. А.Ф. Можайского,*
*197088, Россия, Санкт-Петербург, ул. Ждановская, д.13*

**Аннотация.** Представлены методы и средства реализации программно-управляемого процесса разработки и верификации формальных моделей требований и проектных решений автоматизированных информационных систем критической информационной инфраструктуры. Процессы выполняются в единой для всех его участников модельно-языковой и информационно-программной среде автоматизированным способом на основе предметно-ориентированной онтологий. Онтологии описывают процессы управления качеством программно-технических комплексов на этапах обоснования требований и проектирования систем, разработаны с помощью конструкций и механизмов языков моделирования и проектирования SysML, FUML, OCL, а также математического аппарата сетей Петри, временных автоматов и временных логик. Для валидации и верификации комплекса требований и проектных решений разработаны алгоритмы построения и анализа трассы выполнения модели в среде виртуальной машины VM FUML. Предложены способы интеграции и использования специализированных средств верификации CPN Tools, Rodin, SPIN и Modelica для автоматизированного тестирования моделей комплекса требований и проектных решений. Данный комплекс обеспечивает более эффективное взаимодействие заказчика и исполнителя как при разработке требований, так и при проектировании системы, обнаружение и устранение дефектов посредством реализации автоматизированных процедур верификации, валидации и коррекции. Применение данного подхода позволит повысить качество требований и проектных решений, а также улучшить экономические показатели путем снижения финансовых и временных затрат, связанных с выполнением дополнительных работ как в случае обнаружения дефектов, так и при изменении требований или условий эксплуатации.

**Ключевые слова:** автоматизированные информационные системы; валидация и верификация; временные автоматы; проектирование и моделирование; сети Петри; функциональные и эксплуатационные требования.

**Для цитирования:** Самонов А.В. Методы и средства разработки автоматизированных информационных систем на основе онтологии «Управление качеством программно-технических комплексов». Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 165-182. DOI: 10.15514/ISPRAS-2019-31(5)-13

# Methods and Means for Automated Information Systems Development based on Ontology «Software and Hardware Complexes Quality Management»

*A.V. Samonov, ORCID: 0000-0002-0390-4481 <a.samonov@mail.ru>*

*A.F. Mozhaisky Military Space Academy, Saint-Petersburg, 197198, Russia*

**Abstract.** The paper presents development and verification methods and means of requirements and design solutions formal models. They are intended to create complex critical automated information systems in a same model-language and information-software environment for all its participants. The development and verification processes are carried out in an automated way on the basis of subject-oriented ontologies. Ontologies describe the quality management processes of software and hardware complexes at the stages of requirements justification and system design. They are developing by means modeling and design languages SysML, FUML, OCL structures and mechanisms, the Petri nets mathematical apparatus, time automata and time logics. In order to execute of validation and verification for complex of requirements and design solutions, construction and model execution route analysis algorithms in the VM FUML virtual machine environment are developed. Integration and use methods for specialized verification tools CPN Tools, Rodin, SPIN and Modelica as means to automated testing of complex requirements and design solutions models are proposed. This complex provides more effective interaction between the customer and the contractor both in the development of requirements and in the design of the system, along with this, detection and provides limination of defects through the automated verification, validation and correction procedures implementation. This approach application will improve the quality of requirements and design solutions, as well as improve economic performance by reducing the financial and time costs, which associated with the implementation of additional work in the case of defects, and when changing requirements or operating conditions.

**Keywords:** automated control systems; validation and verification; time machines; design and modeling; Petri nets; functional and operational requirements

**For citation:** Samonov A.V. Methods and Means for Automated Information Systems Development based on Ontology «Software and Hardware Complexes Quality Management». Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019, pp. 165-182 (in Russian). DOI: 10.15514/ISPRAS-2019-31(5)-13

## *1. Введение*

Основными компонентами критической информационной инфраструктуры (КИИ) государства являются информационно-телекоммуникационные сети и автоматизированные информационные системы (АИС), предназначенные для решения задач государственного управления, обеспечения обороноспособности, безопасности и правопорядка. К системам данного класса, являющихся сложными программно-техническими системами (СПТС), предъявляются повышенные требования к надежности, оперативности и устойчивости функционирования. Разработка АИС КИИ представляет собой сложную и ресурсоемкую задачу, результат решения которой не всегда удовлетворяет заданным требованиям и укладывается в рамки выделенных временных и финансовых ресурсов. Данный факт убедительно подтверждают ежегодно публикуемые отчеты американской исследовательской компании The Standish Group. Так, в отчете за 2018 год утверждается, что доля успешных проектов составляет около 30%, 20% проектов оказались проваленными полностью, остальные 50% проектов столкнулись с проблемами, из-за которых итоговый бюджет превысил первоначальный в среднем в 1,5 раза, сроки выросли почти в 2 раза, около 50% функций не соответствовали заявленным требованиям [1].

Одной из основных причин такого состояния дел является отсутствие у участников процесса создания АИС КИИ (пользователя, проектировщика и программиста-разработчика) единой терминологической и концептуальной базы, адекватного логико-

математического аппарата и эффективных средств поддержки процессов формирования и анализа двух важнейших артефактов жизненного цикла (ЖЦ) АИС КИИ: комплекса требований и проектных решений. В данной статьей предложены и описаны модели, методы и средства разработки единой модельно-языковой и информационно-программной среды, а также методика и алгоритмы реализации в ней программно-управляемого процесса разработки и верификации комплекса требований и проектных решений. Реализация данного подхода предполагает разработку и использование онтологии, описывающей процессы управления качеством программно-технических комплексов на различных этапах ЖЦ систем.

## *2. Анализ современных технологий, методов и средств промышленной разработки СПТС. Проблемы и пути решения*

Исключительная актуальность совершенствования технологий и средств разработки надежного программного обеспечения для критически важных автоматизированных систем обусловила огромное внимание и усилия, предпринимаемые международными и национальными организациями, научным и профессиональным сообществом, коллективами разработчиков и отдельными исследователями для решения имеющихся в данной области проблем. Наиболее системными и практичными являются методические документы и спецификации, разработанные под эгидой организации OMG (Object Management Group), с которой сотрудничают около 800 научно-исследовательских организаций (DISA, INCOSE, NIST и др.) и промышленных компаний (AT&T, IBM, Oracle, Microsoft, Cisco Systems, NASA и др.). В настоящее время на сайте OMG опубликовано более 230 методических документов и спецификаций. С точки зрения рассматриваемых здесь вопросов наиболее важными из них являются спецификации: MOF (Meta Object Facility), UML (Unified Modeling Language), XMI (XML Metadata Interchange), SysML (System Modeling Language), OCL (Object Constraint Language), UTP (UML Testing Profile), ALF (Action Language for Foundational UML), FUML (Semantics of a Foundational Subset for Executable UML Models), ReqIF (Requirements Interchange Format).

Теоретической основой современных технологий разработки СПТС являются концепции и методы модельно-ориентированной системной и программной инженерии (МОСиПИ) (Model-based systems engineering, MBSE), которая включает три составных компонента: разработка на основе моделей (Model driven development, MDD), архитектура на основе моделей (Model driven architecture, MDA), тестирование на основе моделей (Model based testing, MBT) [2, 3]. Технологическую основу составляют такие технологии разработки СПТС как Microsoft Solution Framework (MSF), Oracle Method, Rational Unified Process (RUP), SADT (IDEFx). Наиболее известными средствами реализации этих концепций и технологий являются IBM Rhapsody, Sparx Enterprise Arhitect, Modelio, Eclipse Papyrus и некоторые другие. Данные системы обеспечивают разработчиков СПТС автоматизированными средствами анализа, специфицирования, проектирования и тестирования систем, состоящих из аппаратных средств, программного обеспечения, данных, персонала, процедур, средств и других искусственных и природных систем. Для разработки таких средств поддержки используются языки визуального моделирования и проектирования SysML, FUML, OCL, ALF, ArhiMate, AADL. Для контроля качества артефактов ЖЦ систем, получаемых в процессе проектирования и разработки СПТС, применяются различные средства верификации и валидации, например, CPN Tools, Rodin, SPIN, Modelica.

В нашей стране активные исследования в этой области ведутся в таких организациях как ИСП РАН, ВМК МГУ имени М. В. Ломоносова, Санкт-Петербургском ГУ, Новосибирском ГТУ, ВКА имени А.Ф. Можайского и др. В результате этих исследований были созданы: система поддержки проектирования и верификации комплексов бортового

авиационного оборудования MASIW, технология тестирования программных интерфейсов – UniTESK, статический анализатор Svace и др. Ниже представлен краткий обзор основных направлений исследований и работ по развитию и совершенствованию технологий создания СПТС.

Теоретические основы проектирования и верификации СПТС на основе теоретико-категорного похода к метапрограммированию изложены в публикациях ведущего научного сотрудника ИПУ РАН Ковалева С.П. [4, 5]. В них представлены способы применения теории категорий для решения проблемы представления разнородных технологий программной инженерии в единой форме, удобной для их интеграции и координации в рамках общего цикла проектирования программных систем. Особое внимание уделяется современным технологиям, таким, как разработка, управляемая моделями (model checking), и аспектно-ориентированное программирование (aspect-oriented programming), для которых построены универсальные теоретико-категорные семантические модели.

Одним из современных средств описания архитектуры программно-аппаратных систем является Architecture Analysis & Design Language (AADL) [6]. На его основе была разработана и в настоящее время активно используется система поддержки проектирования и верификации комплексов бортового авиационного оборудования MASIW, разработанная ИСП РАН совместно с ГосНИИАС в рамках государственной программы по развитию Интегрированной Модульной Авионики (ИМА). При разработке MASIW были использованы библиотеки и средства Eclipse Modeling Framework, Graphical Editing Framework, Eclipse Team Providing, SVN Team Provider, GIT Team Provider. Как отмечено в статье [7], инструментальные средства MASIW позволяют решать следующие задачи:

- создание, редактирование и управление моделями программно-аппаратных комплексов (ПАК) на языке AADL;
- анализ моделей на предмет достаточности аппаратных ресурсов и согласованности интерфейсов, оценки характеристик проектируемых сетей передачи данных, построенных в соответствии со стандартом AFDX (Avionics Full-Duplex Switched Ethernet);
- распределение функциональных приложений по вычислительным модулям с учетом ограничений ресурсов аппаратной платформы и требований к надежности и безопасности ПАК;
- генерация программного кода и конфигурационных данных для ОС РВ VxWorks653 и оконечных устройств сети AFDX.

Пример использования специального расширения языка AADL – Error Model Annex (EMA) и инструмента MASIW для моделирования и анализа безопасности проектируемых ПАК представлен в статье [8]. С помощью EMA создается модель, в которой для каждого компонента ПАК разрабатывается конечный автомат, состояниями которого являются штатные и нештатные, в том числе опасные и отказные состояния данного компонента. Влияние сбоев компонентов системы на другие компоненты описывается посредством указания логических условий распространения ошибок между различными типами компонентов в различных состояниях с учетом вероятностей их возникновения.

Для анализа рисков используются следующие алгоритмы: анализ дерева неисправностей, анализ видов и последствий отказов, марковский анализ. Реализация описанного в данной статье подхода позволяет выявлять и устранять критичные для безопасности создаваемой системы дефекты проектных решений уже на этапе ее проектирования.

В статье [9] представлен обзор следующих методов автоматической генерации тестов для верификации программного обеспечения:

- структурного тестирования с помощью символического выполнения (structural testing using symbolic execution);
- тестирования на основе моделей (model-based testing);
- комбинаторного тестирования (combinatorial testing);
- выборочного тестирования (random testing);
- поиска на основе тестирования (search-based testing).

В статье [10] представлен автоматизированный способ построения UML диаграмм последовательностей из описания UML диаграмм вариантов использования и диаграмм классов. Для его реализации используется язык ATL и разработанные авторами статьи метамодели диаграмм вариантов использования, классов и последовательностей, а также правила получения из первых двух диаграмм третьей. Результатом преобразования является диаграмма последовательностей в формате XMI, которая затем преобразуется в формат XSLT для отображения диаграммы последовательностей в среде графического редактора для просмотра, анализа и доработки. Недостатком предложенного алгоритма является отсутствие возможности автоматической коррекции исходных моделей в случае внесения изменений в диаграмму последовательностей. Это обусловлено тем, что преобразования с помощью языка ATL являются однонаправленными, т.е. работают с моделями источника только для чтения и создают целевые модели только для записи.

В статье [11] описан метод автоматической генерации программного кода на основе проекта программы, представленного на языке ALF. Особого внимания заслуживает описание концептуальной схемы механизма генерации программного кода с помощью средств редактора ALF и используемых для этого правил в нотации расширенной формы Бэкуса-Наура (Enhanced-BackusNaur-Form (EBNF) – язык ATL). Авторы отмечают следующие достоинства инструмента трансформации модели архитектуры языка ATL: возможность описания как декларативных, так и императивных языковых конструкций, наличие средств объединения модулей, позволяющих создавать и повторно использовать наборы правил преобразования. Результатом является программный код на языке Java, соответствующий метамодели Modisco Java.

В статье [12] дано описание двух методов реализации автоматической проверки нагруженных систем реального времени с использованием сценариев. В первом система моделируется как сеть временных автоматов (ВА). Во втором – как набор диаграмм динамических последовательностей (LSCs) и требований в виде отдельной анализируемой диаграммы LSC. Авторы статьи разработали временные (темпоральные) расширения для подмножества ядра языка LSC и определили его семантику на основе трассировки. Анализируемая диаграмма LSC транслируется в ее поведенческий эквивалент в нотации диаграммы ВА. Верификация корректности модели осуществляется посредством моделирования диаграммы ВА в режиме реального времени с использованием аппарата темпоральной логики CTL (Computational Tree Logic) с последующим сравнением полученного результата с эталоном. Оба метода реализованы с помощью средств инструмента UPPAAL.

В работе [13] предлагается метод построения разверток для безопасных консервативных вложенных сетей Петри, основанный на трансляции таких сетей в классические сети Петри. Для классических сетей Петри затем применяются стандартные методы построения разверток. Также в работе обсуждаются сравнительные достоинства двух подходов.

Одним из первых этапов разработки СПТС является их декомпозиция на функциональные компоненты. Исследованию и совершенствованию методов декомпозиции программных систем посвящено множество работ.

В статье [14] представлены два подхода решения данной задачи, которые были использованы при дедуктивной верификации формальной спецификации мандатной

сущностно-ролевой модели управления доступом и информационными потоками в ОС семейства Linux (МРОСЛ ДП-модель) - с использованием формального метода Event-B и техники пошагового уточнения.

В статье [15] предложены методические рекомендации и комплекс средств реализации процесса сквозного контроля качества СПТС на всех этапах их жизненного цикла: обоснования и формализации требований, проектирования, реализации, испытаний. Для реализации данного процесса разработана единая модельно-языковая и информационно-программная среда, основанная на онтологии автоматизируемой предметной области, использующая языки визуального проектирования и моделирования fUML, OCL, ALF, а также автоматизированные средства валидации и верификации CPN Tools, Rodin, SPIN.

В настоящее время ведутся активные исследования и работы по созданию предметно-ориентированных языков моделирования (xDSML, eXecutable Domain-Specific Modeling Languages), использующие и развивающие ключевую концепцию МОСиПИ – метамоделирование. Результатом таких работ, в частности, является разработанный группой исследователей из Франции и Германии программный комплекс GEMOC Studio [16]. Данный комплекс предназначен для создания xDSML на основе языков моделирования SysML/FUML, разработки исполняемых моделей на этом языке (xDSML), их валидации и верификации с помощью графического аниматора и интеллектуального механизма трассировки. С помощью данного комплекса можно осуществлять мониторинг состояния анализируемых моделей (переходов, событий, значений переменных) во время их выполнения в виртуальной машине.

В работе [17] дано описание используемой в данном средстве многомерной предметно-ориентированной метамодели трассировки, обеспечивающей более высокую производительность по сравнению со стандартной UML метамоделью за счет исключения из обработки избыточных данных.

В работе [18] для создания средств динамической верификации и валидации проектных поведенческих моделей предлагается использовать исполняемые предметно-ориентированные языки моделирования xDSMLs (Executable Domain-Specific Modeling Languages). Средства на их основе позволяют осуществлять мониторинг состояния анализируемых моделей (переходов, событий, значений переменных) во время их выполнения. Предложен новый генеративный подход, основанный на многомерной предметно-ориентированной метамодели трассировки, с помощью которого реализуется построение и управление трассами исполнения для моделей, соответствующих заданному xDSML. Как утверждают авторы работы, данный метод имеет более высокую производительность по сравнению со стандартной UML метамоделью, благодаря возможности исключать из обработки избыточные данные (например, анализируемые трассы) с помощью механизмов соответствующего xDSML.

Завершая анализ публикаций и представленных в них решений, можно сделать следующие выводы:

- основные усилия исследователей направлены на разработку методов и средств автоматизированной генерации и верификации программных реализаций СПТС, в меньшей степени – на автоматизацию разработки и верификации проектных решений и практически отсутствуют решения для автоматизированного формирования и верификации комплекса требований;
- в качестве основных математических моделей и построенных на их основе средств для автоматической верификации используются математический аппарат и алгоритмы анализа сетей Петри, темпоральные логики, исполняемые FUML и xUML модели, SMT/SAT решатели задач, а также языки моделирования AADL, UML, FUML, SysML, OCL и разработанные на их основе предметно-ориентированные языки xDSMLs.

Таким образом, следуя принципам и методологии модельно-ориентированного подхода, можно утверждать, что первоочередными задачами, которые необходимо решить для реализации программно-управляемого процесса разработки АИС КИИ, являются:

- построение единой модельно-языковой и информационно-программной среды разработки и верификации АИС КИИ;
- разработка алгоритмического, инструментального и методического обеспечения реализации программно-управляемого процесса разработки и верификации формальных моделей требований, архитектуры и реализации АИС КИИ.

Для построения единой модельно-языковой и информационно-программной среды разработки и верификации АИС КИИ предлагается использовать:

- языки моделирования UML, FUML, SysML, OCL и ALF;
- методы и средства разработки предметно-ориентированной онтологии АИС КИИ (ODM – Ontology Definition Metamodel, RDF – Resource Description Framework);
- библиотеки и программные продукты, реализованные в рамках проекта Eclipse: Eclipse Modeling Framework, Graphical Editing Framework, Papyrus, Modelio.

Выбор этих моделей, языков и средств обусловлен тем, что, во-первых, их развитие активно поддерживается со стороны ведущих предприятий разработчиков и организаций потребителей СПТС, во-вторых, как сами технологии, так и основанные на них средства являются открытыми и доступными для изучения, применения и совершенствования.

## *3. Модели, методы и средства построения единой модельно-языковой и информационно-программной среды для разработки комплекса требований и проектных решений*

На основе проведенного анализа нормативно-методических документов, научно-технических публикаций и программных средств проектирования и разработки АИС КИИ для построения единой модельно-языковой и информационно-программной среды были выбраны следующие конструкции и механизмы языков визуального проектирования SysML, FUML и OCL:

- диаграммы пакетов (Package), внешнего и внутреннего представления блоков (Block Definition, Internal Block), активных и пассивных классов (Active Class, Passive Class) - для описания состава и структуры системы;
- диаграммы вариантов использования (UseCase), деятельности (Activity), состояний (State Machine) и взаимодействия (Sequience) - для описания поведения и способов взаимодействия ее компонентов;
- средства языка OCL – для описания требований к эксплуатационным характеристикам (производительности, надежности, защищенности и др.);
- отношения обобщения (generalization), агрегации (aggregation), композиции (composition), ассоциации (association), зависимости (dependency), представленные с помощью средств языка OCL - для описания связей между структурными, поведенческими, ограничительными и другими аспектами, элементами и свойствами системы.

Для валидации и верификации формальных моделей комплекса требований и проектных решений целесообразно использовать виртуальную машину *VM FUML*, а также такие инструменты как *CPN Tools* [19], *Rodin* [20], *SPIN* [21, 22]. На рис. 1 отражена предлагаемая схема (алгоритм) построения формальных моделей комплекса требований и проектных решений АИС КИИ.

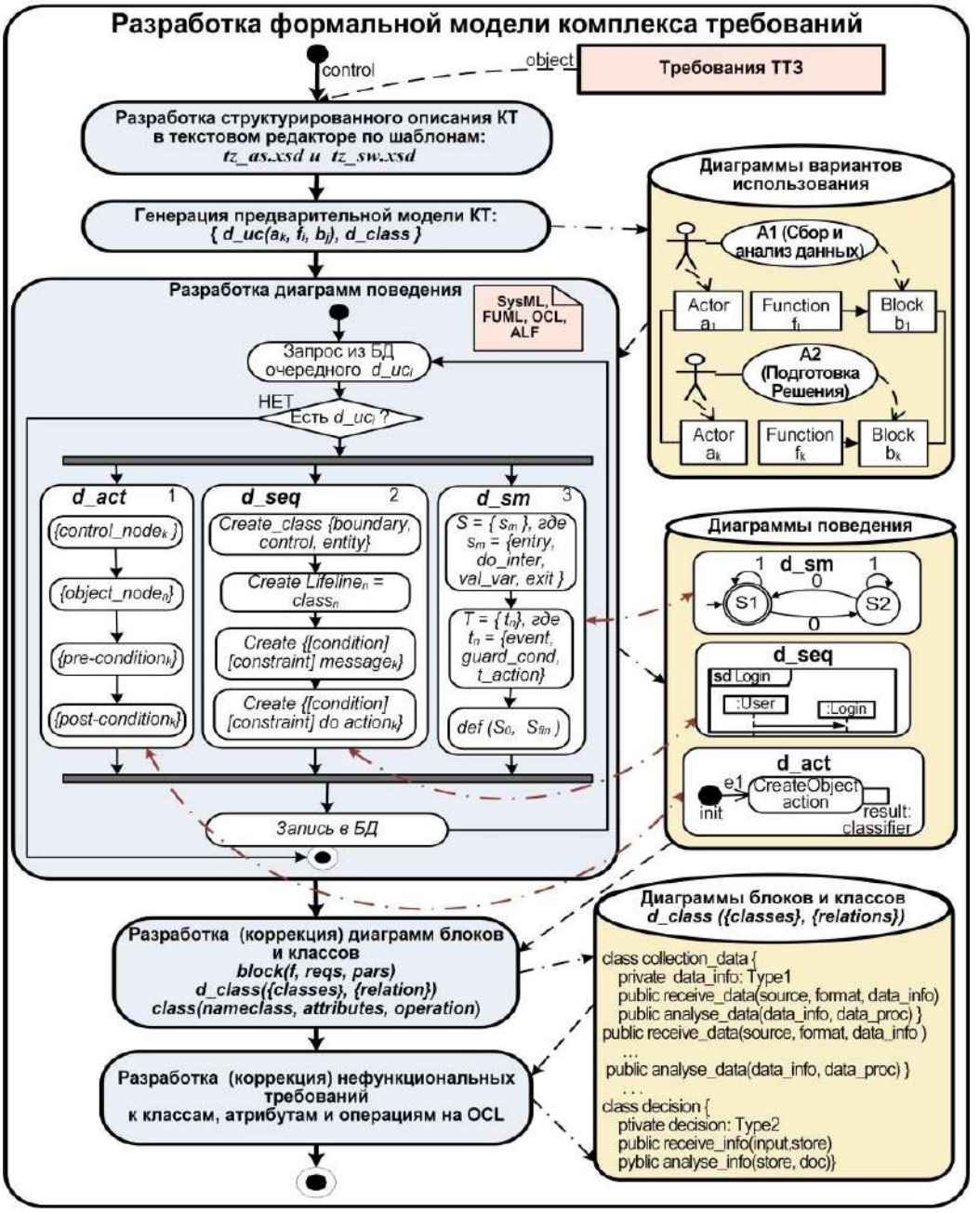


*Рис. 1. Схема построения формальных моделей комплекса требований и проектных решений АИС КИИ*

*Fig. 1. Scheme for formal model construction of requirements and design complex for automated information systems*

Построение формальных моделей требований и проектных решений в соответствии с выше представленной схемой заключается в выполнении следующих процедур:

- разработки структурированного описания комплекса требований (КТ) в текстовом редакторе на основе специальных шаблонов: tz_as.xsd и tz_sw.xsd;

- генерации предварительной модели КТ в виде совокупности предметно-ориентированных диаграмм вариантов использования;
- построения для каждой такой диаграммы диаграмм поведения (d_act, d_seq, d_sm), обеспечивающих более подробное и точное описание требований к порядку, способам и условиям реализации функционала системы;
- описания состава и структуры системы в виде совокупности диаграмм специализированных блоков и классов (d_blocks, d_class);
- определения нефункциональных (эксплуатационно-технических) требований к качеству реализации функционала, среде и условиям эксплуатации АИС.

В следующих разделах статьи описаны модели и средства реализации процедур автоматизированной разработки и верификации формальных моделей требований и проектных решений.

## *4. Методы и средства разработки и верификации формальной модели комплекса требований к АИС КИИ*

Управление качеством АИС КИИ должно осуществляться на всех этапах их ЖЦ и представлять собой тесно связанную совокупность следующих процессов:

1) выбор и обоснование характеристик и показателей характеристик качества (ПХК), к которым следует предъявлять требования, и определение критериев, которым значения ПХК должны удовлетворять;
2) обеспечение требований к качеству АИС КИИ на этапе проектирования;
3) обеспечение требований к качеству АИС КИИ на этапе реализации;
4) измерение и оценивание ПХК на этапе приемо-сдаточных испытаний;
5) обеспечение требований к качеству АИС КИИ на этапе эксплуатации и сопровождения.

Для каждого из этих процессов необходимо разработать соответствующую онтологию. В данной статье основное внимание уделяется первым двум процессам и используемым для их реализации онтологиям.

Комплекс требований к АИС КИИ включает две основные группы требований: требования к ее функциональным возможностям и требования к качеству их реализации.

Для обеспечения автоматизированного построения формальных описаний требований в среде визуального моделирования были разработаны предметно-ориентированные шаблоны (стереотипы), представляющие собой расширения диаграмм вариантов использования (*UseCase*), деятельности (*Activity*), состояний (*State Machine*), взаимодействия (*Sequence*), внешнего (*block definition diagram*) и внутреннего (*internal block diagram*) представления компонентов системы, активных (*Active Class*) и пассивных (*Passive Class*) классов. Каждый *i*-й вариант использования представляет собой функциональное требование и описывается следующим образом:

$$uc_i = (nameuc_i,\ description_i,\ actor_k, subject_j,\ basic\_scenario_i,\ alter\_scenario_i).$$

Функциональные блоки системы (*subject*) описываются посредством соответствующих расширений диаграмм пакетов (*Package*), внешнего и внутреннего представления блоков (*Block Definition, Internal Block*). Активные классы (*Active Class*) описывают пользователей, системы и элементы внешнего окружения, которые могут инициировать какую-либо деятельность АИС.

Пассивные классы (*Passive Class*) описывают артефакты, создаваемые и используемые АИС во время функционирования (информационные, материальные, вычислительные, сетевые, людские ресурсы; решения, команды, сигналы, события, …).

$Active\ Class = < Actors, Systems, Envs >.$

Основной и альтернативные сценарии реализации функций представляются посредством специализированных активных и пассивных классов (*Active Class, Passive Class*). Для описания методов этих классов используются предметно-ориентированные диаграммы поведения, построенные с помощью механизма расширения из диаграмм деятельности (*Activity*), состояний (*State Machine*) и взаимодействия (*Sequence*).

Все АИС КИИ, с точки зрения особенностей алгоритма их функционирования, можно разделить на три группы. Первую группу образуют системы, выполняющие свои функции в соответствии с определенным алгоритмом, имеющим вход и выход. Во вторую группу входят реагирующие системы, выполняющие определенные операции и изменяющие свое состояние в зависимости от внешних воздействий. Третью группы образуют системы, взаимодействующие посредством обмена сообщениями.

Для формального описания функционирования систем первой группы наиболее адекватным средством являются диаграммы деятельности (*activity*), основанные на математическом аппарате сетей Петри (СП).

Классическая СП описывается следующей формулой:

$$CPN = (P, T, F, m_I),$$

где $P = \{p_1, p_2, \dots, p_n\}$ – множество мест;

$T = \{t_1, t_2, \dots, t_n\}$ – множество переходов, таких что $P \cap T = \emptyset$;

$F \subseteq P \times T \cup T \times P$ – матрица инцидентности;

$m_I : P \to N$ – начальная маркировка.

Основными свойствами СП, которые следует использовать для верификации формальных моделей требований и архитектуры, являются: живость и ограниченность. СП обладает свойством живости, если она обладает свойствами достижимости и покрываемости. Достижимость – это наличие путей перехода в заданные состояния из начального состояния $m \in [m_I\rangle$. Покрываемость – возможность перехода в заданные состояния из других состояний $m'' \in [m'\rangle$. Сеть Петри является $k-$ограниченной, если и только если все маркировки $m \in [m_I\rangle$, и для всех $p \in P$: $m(p) \leq k$. Обладающая данным свойством СП и, как следствие, описываемая с ее помощью система будет функционировать в определенных границах. Частным случаем свойства ограниченности является безопасность. Сеть Петри $(P, T, F, m_I)$ является безопасной, если все маркировки $m \in [m_I\rangle$, и для всех $p \in P$: $m(p) \leq 1$.

Важным достоинством использования СП является наличие целого ряда автоматизированных средств верификации построенных с их помощью моделей. Одним из наиболее развитых и эффективных является *CPN Tools*, который и предлагается использовать в нашем случае для анализа формальных моделей требований и архитектуры АИС КИИ.

Формальная модель комплекса требований подвергается процедурам валидации и верификации. Процедура валидации заключается в оценивании комплекса требований на предмет его полноты и корректности и выполняется как программными средствами, так и неформальной экспертизой специалистов в данной предметной области.

При верификации проверяются такие свойства комплекса требований, как непротиворечивость, системность, неизбыточность, безопасность, живость, отсутствие взаимоблокировок, невыполнимых операций, зацикливаний.

Следующим шагом построения модели комплекса требований является разработка требований к качеству реализации каждой функции. Наиболее важными эксплуатационным характеристикам АИС КИИ являются: надежность, защищенность и производительность. Для реализации программно-управляемого процесса их описания разработана онтология «Требования к качеству реализации функциональных возможностей АИС КИИ» (рис.2).

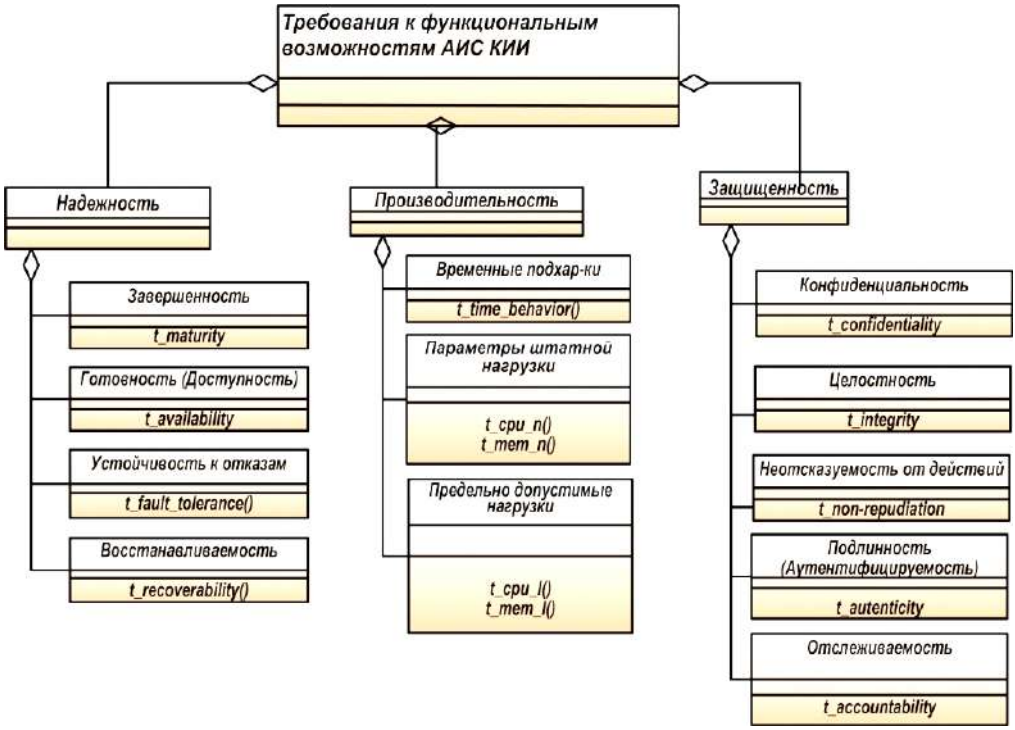


*Рис. 2. Онтология «Требования к качеству реализации функциональных возможностей АИС КИИ»*
*Fig. 2. «Requirements to Realization Quality of Functional Abilities for automated information systems» Ontology*

Для задания требований к характеристике качества «надежность» используются следующие ПХК и шаблоны (функции): *t_maturity* – для ПХК «завершенность», *v_fault tolerance* – для ПХК «отказоустойчивость», *v_recoverability* – для ПХК «восстанавливаемость», *t_availability* – для ПХК «готовность». Данные ПХК определяют требования к качеству реализации соответствующих функциональных возможностей АИС КИИ.

Для задания требований к характеристике качества «производительность» предлагается использовать характеристики времени выполнения функций в различных режимах работы, которые определяются такими параметрами как количество пользователей, объемы и темп запросов, сложность и ресурсоемкость алгоритмов решения задач и др. При этом необходимо проанализировать как минимум два режима: штатной и максимальной нагрузки.

Требования к эксплуатационным характеристикам (производительности, надежности, защищенности, совместимости и др.) описываются с помощью средств языка *OCL*. Для каждой из этих характеристик разработаны соответствующие шаблоны, которые используются в подключаемых к инструменту *Eclipse Papyrus* модулях (*plugins*) для реализации программно-управляемого процесса их формального описания и

верификации. Затем с помощью отношений обобщения (*generalization*), агрегации (*aggregation*), композиции (*composition*), зависимости (*dependency*), подчиненности (*subordinations*), дислокации (*dislocations*) и др. устанавливаются связи между структурными, поведенческими, ограничительными и другими аспектами и характеристиками модели, которые также представляются с помощью средств языка OCL.

Состав и структура комплекса моделей и программных средств, с помощью которых реализуются процедуры верификации и валидации формальной модели требований к АИС КИИ, представлены на рис. 3.

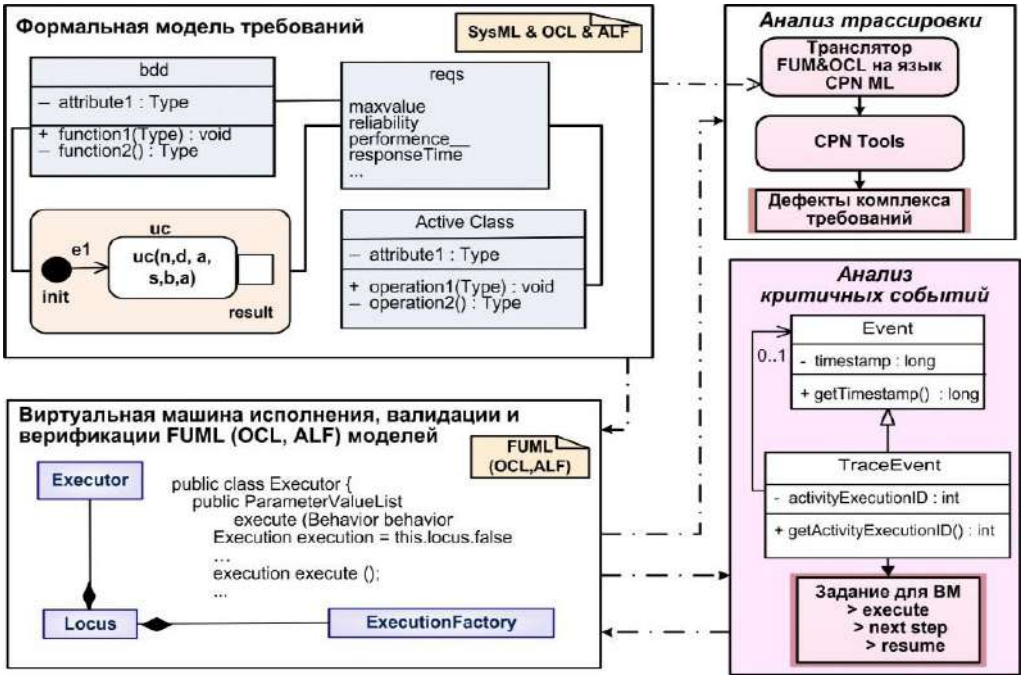


*Рис. 3. Состав и структура комплекса моделей и программных средств реализации процедур верификации и валидации формальной модели требований к АИС КИИ»*
*Fig. 3. Set and Structure of Complex Models and Software for Formal Requirements Model Validation and Verification*

Процедура тестирования комплекса требований включает:

- валидацию комплекса требований посредством исполнения его формальной модели в среде виртуальной машины VM FUML и анализа ее состояния в контрольных точках и при обработке критичных событий;
- анализ поведенческой части модели, представленной диаграммами активности, с помощью *CPN Tools* на предмет выявления тупиковых и мертвых состояний, стабильности, ограниченности и безопасности;
- анализ поведенческой части модели, представленной диаграммами состояний, с помощью инструмента *Rodin* на предмет выявления тупиковых и мертвых состояний, необрабатываемых стимулов и др.;
- верификацию качества реализации в диаграмме классов базовых механизмов объектно-ориентированной парадигмы (инкапсуляции, наследования, полиморфизма, абстракции, обмена сообщениями) посредством расчета и оценивания метрик, характеризующих эти механизмы.

## *5. Методы и средства разработки и верификации архитектуры и проектных решений*

Проектирование архитектуры и проектных решений АИС КИИ осуществляется системным архитектором на основе разработанного и верифицированного на предыдущем этапе комплекса требований в той же модельно-языковой и информационно-программной среде.

Для этого ему предоставляются графические и табличные шаблоны проектирования структурных и поведенческих аспектов системы.

При разработке шаблонов для проектирования архитектуры и проектных решений АИС КИИ с помощью диаграмм активности были использованы раскрашенные временны*е* сети Петри (РВСП), формальное описание которых представляется следующим выражением:

$$TPN \equiv < P, T, C, F, m_I, S, Z >,$$

где $P$, $T$, $F$ *и* $m_i$– имеют тоже значение, что и для простых сетей Петри;

$C=\{c_1, c_2, \ldots, c_d\}$ – цвета или типы маркеров;

$S=(s_1, s_2, \ldots, s_n)$ – вектор временных задержек маркеров в позициях;

$Z = \{z_1, z_2, \ldots, z_r\}$ – вектор времени срабатывания разрешенных переходов.

При разработке шаблонов для проектирования архитектуры и проектных решений АИС КИИ с помощью диаграмм состояний были использованы математические модели временных автоматов (ВА) [23, 24], описываемые следующим выражением:

$TA \equiv < S, C, D, R, f, s_0 >$, где:

$S$ – конечное множество позиций автомата;

$C$ – конечное множество локальных часов, значения которых возрастают синхронно с реальным временем и могут принимать значения из множества действительных чисел $\Re$;

$D$ – конечное множество действий;

$R \subseteq S \times P \times D \times C \times S$ - множество переходов автомата;

$f: S \to P$ – функция, которая ставит в соответствие каждой позиции $s \in S$ некоторый предикат $p \in P$ ;

$s_0 \in S$ – начальная позиция автомата.

Формальная модель архитектуры и проектных решений должна удовлетворять следующим характеристикам качества:

- полнотой и корректностью реализации функциональных требований;
- полнотой и корректностью реализации эксплуатационных требований;
- согласованностью и непротиворечивостью всех диаграмм модели;
- отсутствием избыточности диаграмм и их атрибутов;
- отсутствием взаимоблокировок, невыполнимых операций и зацикливаний, которые могут привести к нарушению безопасности и живости системы.

Для разработки программных средств верификации проектных решений была разработана и использована онтология «Оценивание качества реализации АИС КИИ» (рис. 4).

Все требования к качеству АИС в данной модели сведены в 8 групп: функциональная пригодность, производительность, надежность, совместимость, удобство использования, удобство сопровождения, защищенность, переносимость. Функциональная пригодность включает три показателя качества характеристики (ПХК): корректность, полноту и целесообразность. Для расчета и оценивания

соответствия этих ПХК заданным требованиям используются функции: *v_functional_completeness*, *v_functional_correctness* и *v_functional_appropriateness*.

Для расчета и оценивания характеристики качества «надежность» используются следующие функции: *v_maturity* – для ПХК «завершенность», *v_availability* – для ПХК «готовность», *v_fault tolerance* – для ПХК «отказоустойчивость», *v_recoverability* – для ПХК «восстанавливаемость».

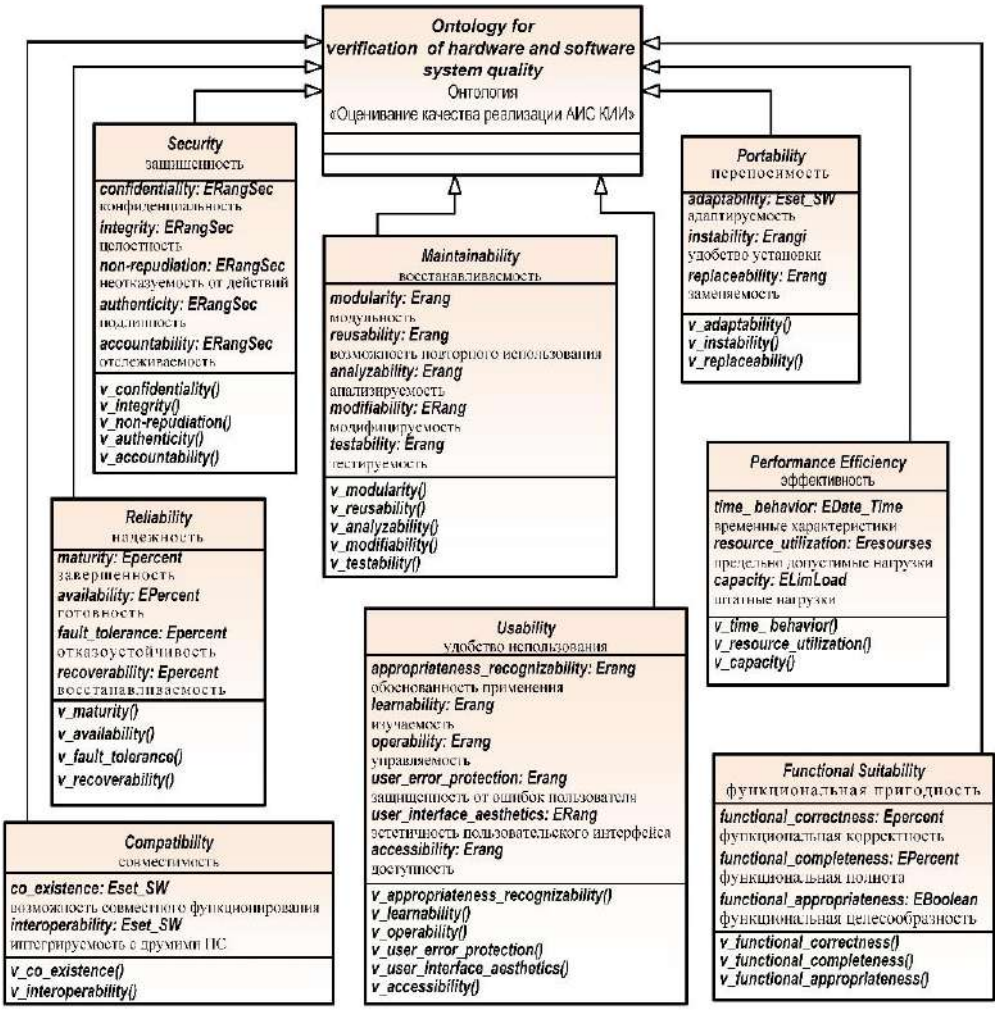


*Рис. 4. Онтология «Оценивание качества реализации АИС КИИ»*
*Fig. 4. «Verification of hardware and software System Realization Quality» Ontology*

Для оценивания характеристики «производительность» используются характеристики времени выполнения функций в двух основных режимах работы: штатной и максимальной нагрузки. При расчете должны учитываться доступные вычислительные, программные, сетевые и другие ресурсы.

Состав и структура комплекса программных средств, обеспечивающих проверку соответствия характеристик качества проекта АИС КИИ заданным требованиям, представлены на рис. 5.

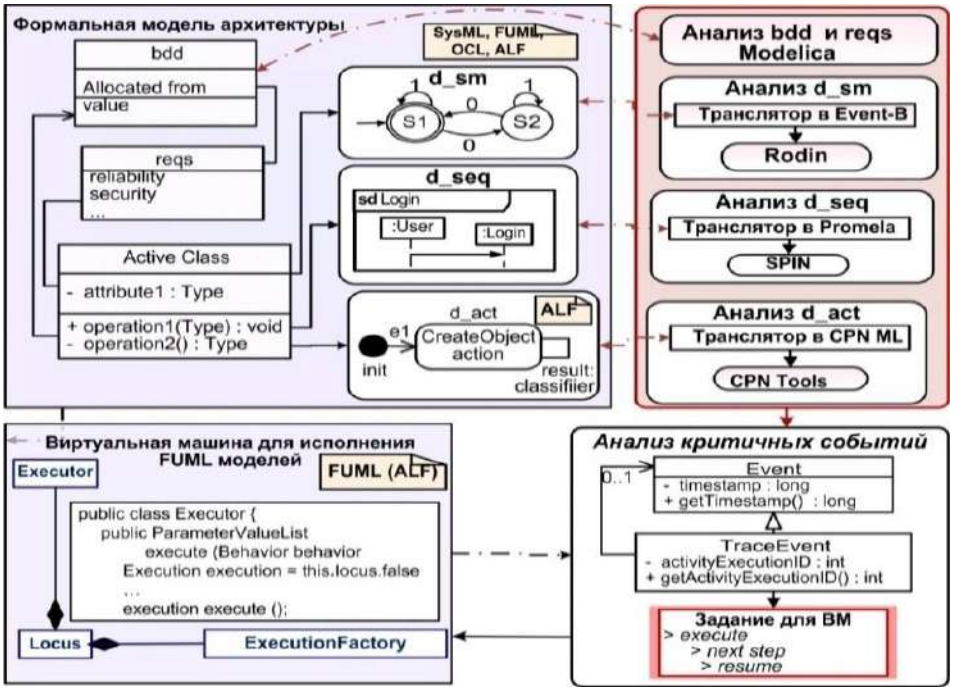


*Рис. 5. Состав и структура комплекса программных средств, используемых для валидации и верификации проектных решений*
*Fig. 5. Set and Structure of Complex Software for Design Decisions, Validation and Verification*

Для верификации технической части проектных решений предлагается использовать средство моделирования кибер-физических систем *OpenModelica* [25]. В случае обнаружения каких-либо дефектов проектные решения возвращаются архитекторам системы на доработку. Исправленный проект подвергается повторной процедуре валидации и верификации. В результате реализации данного последовательно-итерационного и программно-управляемого процесса будут разработаны проектные решения, в полной мере соответствующие предъявленным к АИС КИИ требованиям пользователей, условиям применения и нормативным документам.

## *6. Заключение*

Представленные в статье методы и комплексы программных средств предназначены для реализации программно-управляемого процесса разработки, верификации и валидации АИС КИИ. Работа на всех этапах ЖЦ осуществляется в единой модельно-языковой и информационно-программной среде, построенной на основе технологий объектно-ориентированного анализа и проектирования, языков и средств визуального моделирования SysML, FUML, OCL. Их применение сокращает логические и семантические разрывы между представлениями о разрабатываемой системе у участников процесса создания АИС КИИ, существенно повышая эффективность их взаимодействия. Это приведет к повышению качества всех артефактов жизненного цикла разработки систем, и в первую очередь комплекса требований и проектных решений.

Реализация автоматизированных процедур верификации, валидации и коррекции комплекса требований и проектных решений с помощью таких средств как VM FUML, CPN Tools, SPIN и Rodin, позволит улучшить экономические показатели в части снижения

финансовых и временных затрат, связанных с выполнением дополнительных работ по внесению изменений, как в случае обнаружения каких-либо дефектов, так и при изменении самих требований или условий эксплуатации АИС КИИ.

## Список литературы / References


[1]. The Standish Group report. URL: https://www.standishgroup.com/store/services/10-chaos-report-decision-latency-theory-2018-package.html, accessed 21.05.2019.

[2]. Systems Engineering and Software Engineering [online]. URL: https://www.sebokwiki.org/wiki/Systems_Engineering_and_Software_Engineering, accessed 25.05.2019.

[3]. Laura E. Hart. Introduction To Model-Based System Engineering (MBSE) and SysML [online]. URL: https://www.incose.org/docs/default-source/delaware-valley/mbse-overview-incose-30-july-2015.pdf, accessed 21.05.2019.

[4]. Ковалёв С.П. Теоретико-категорный подход к метапрограммированию. М.: ИПУ РАН, 2014, 112 стр./ S.P. Kovalev. Category-theoretic approach to metaprogramming. M.: ICS RAS, 2014, 112 p. (in Russian).

[5]. Ковалев С.П. Теоретико-категорный подход к проектированию программных систем. Фундаментальная и прикладная математика, том 19, вып. 3, 2014 г., стр. 111–170 / Kovalev S.P. Category-theoretic approach to the design of software systems. Fundamental and Applied Mathematics, vol. 19, issue 3, 2014, pp. 111-170 (in Russian).

[6]. Peter H. Feiler, David P. Gluch. Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. Addison-Wesley Professional, 2012, 479 p.

[7]. 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 / D.V. Buzdalov, S.V. Zelenov, E.V. Kornykhin, A.K. Petrenko, A.V. Strakh, A.A. Ugnenko, A.V. Khoroshilov. Design tools for integrated modular avionics systems. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 1, 2014, pp. 201-230 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-6.

[8]. Зеленов С.В., Зеленова С.А. Моделирование программно-аппаратных систем и анализ их безопасности. Труды ИСП РАН, том 29, вып. 5, 2017 г., стр. 257-282 / S.V Zelenov, S.A. Zelenova Modeling of software and hardware systems and analysis of their safety. Trudy ISP RAN/Proc. ISP RAS, vol. 29, vol. 5, 2017, pp. 257-282 (in Russian). DOI: 10.15514/ISPRAS-2017-29(5)-13

[9]. An Orchestrated Survey on Automated Software Test Case Generation. Journal of Systems and Software, vol. 86, issue 8, 2013, pp. 1978-2001.

[10]. Photchana Sawprakhon, Yachai Limpiyakorn. Sequence Diagram Generation with Model Transformation Technology. In Proc. of the International MultiConference of Engineers and Computer Scientists, 2014, pp. 584-589.

[11]. Thomas Buchmann and Alexander Rimer. Unifying Modeling and Programming with ALF. In Proc. of the Second International Conference on Advances and Trends in Software Engineering, 2016, pp. 10-15.

[12]. Li S., Balaguer S., David A. et al. Scenario-based verification of real-time systems using Uppaal. Formal Methods in System Design, vol. 37, Issue 2–3, 2010, pp 200–264.

[13]. Ермакова В.О., Ломазова И.А. Трансляция вложенных сетей Петри в классические сети Петри для верификации разверток. Труды ИСП РАН, том 28, вып, 4, 2016, стр. 115-136 / Ermakova V.O., Lomazova I.A. Translation of Nested Petri Nets into Petri Nets for Unfoldings Verification. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 4, 2016, pp. 115-136 (in Russian). DOI: 10.15514/ISPRAS-2016-28(4)-7.

[14]. Девянин П.Н., Кулямин В.В., Петренко А.К., Хорошилов А.В., Щепетков И.В. Сравнение способов декомпозиции спецификаций на Event-B. Программирование, том 42, № 4, 2016 г., стр. 17-26 / Comparison of specification decomposition methods in Event-B Devyanin P.N., Kulyamin V.V., Petrenko A.K., Khoroshilov A.V., Shchepetkov I.V. Programming and Computer Software, vol. 42, № 4, 2016, pp. 198-205.

[15]. Самонов А.В., Самонова Г.Н. Методика и средства разработки и верификации формальных FUML моделей требований и архитектуры сложных программно-технических систем. Труды ИСП РАН, том 30, вып. 5, 2018, стр. 125-148 / Samonov A.V., Samonova G.N. Methodology and Tools for Development and Verification of formal fUML models of Requirements and Architecture

for complex software and hardware systems. Trudy ISP RAN/Proc. ISP RAS, 2018, vol. 30, issue 5, pp.125-148. DOI: 10.15514/ISPRAS-2018-30(5)-8.

[16]. Eclipse GEMOC Studio [online]. URL: https://projects.eclipse.org/projects/modeling.gemoc, accessed 20.06 2019.

[17]. Meyers B., Deshayes R., Lucio L., Syriani E., Vangheluwe H., Wimmer M. ProMoBox: A Framework for Generating Domain-Specific Property Languages. Lecture Notes in Computer Science, vol. 8706, 2014, pp. 1-20.

[18]. Bousse E., Mayerhofer T., Combemale B., Baudry B. Advanced and efficient execution trace management for executable domain-specific modeling languages. Software & Systems Modeling, vol. 18, issue 1, 2019, pp 385–421.

[19]. CPN Tools Homepage. CPN Tools is a tool for editing, simulating, and analyzing Colored Petri nets URL: http://cpntools.org/, accessed 27.05.2019.

[20]. Rodin. URL: http://www.event-b.org/, accessed 11.05.2019.

[21]. SPIN. URL: http://spinroot.com/spin/whatispin.html, accessed 11.06.2019.

[22]. Ю.Г. Карпов, И.В. Шомшина. Введение в язык Promela и систему комплексной верификации Spin: учебное пособие. Изд-во Политехнического ун-та, 2010, 110 стр. / Y.G. Karpov, I.V. Shamshina. Introduction to the language Promela and the verification system of the comprehensive Spin: a training manual. Publishing house of Polytechnic University, 2010, 110 p. (in Russian).

[23]. Timed Automata: Semantics, Algorithms and Tools. Johan Bengtsson and Wang Yi. Lecture Notes in Computer Science, vol. 3098, 2003, pp. 87-124.

[24]. Твардовский А.С., Лапутенко А.В. О возможностях автоматного описания параллельной композиции временных автоматов. Труды ИСП РАН, том 30, вып. 1, 2018 г., стр. 25-40. / A.S Twardowski., AV. Lopotenco. On the possibilities of automaton description of parallel composition of time automata. Trudy ISP RAN/Proc. ISP RAS, volume 30, vol. 1, 2018, pp. 25-40 (in Russian). DOI: 10.15514/ISPRAS-2018-30(1)-2.

[25]. Openmodelica. URL: https://www.openmodelica. org/, accessed 27.05.2019.

## Информация об авторе / Information about author

Александр Валерьянович САМОНОВ – кандидат технических наук, доцент, старший научный сотрудник. Сфера научных интересов: системный анализ, информатика, математическое и имитационное моделирование и разработка сложных программно-технических систем, методы и средства обеспечения информационной безопасности.

Alexander Valerianovitch SAMONOV – PhD in Technical Sciences, Associated Professor, Senior Researcher. Research interests: system analysis, computer science, system and software engineering, methods and means of information security.

**DOI: 10.15514/ISPRAS-2019-31(5)-14**



# Динамическое построение прогноза времени завершения вычислительного эксперимента в Desktop Grid

[1,2] *Е.Е. Ивашко, ORCID: 0000-0001-9194-3976 <ivashko@krc.karelia.ru>*
[2] *В.С. Литовченко, ORCID: 0000-0003-1104-0502 <valentina97@yandex.ru>*

[1] *Институт прикладных математических исследований*
*Карельского научного центра РАН,*
*185910, Россия, г. Петрозаводск, Пушкинская, д. 11*
[2] *Петрозаводский государственный университет,*
*185910, Россия, г. Петрозаводск, Ленина, д. 33*

**Аннотация.** Desktop Grid является мощным инструментом высокопроизводительных вычислений. Desktop Grid – это форма распределенной высокопроизводительной вычислительной системы, которая использует время простоя географически распределенных вычислительных узлов, объединенных низкоскоростной сетью. При этом, системы типа Desktop Grid имеют существенные отличия от традиционных вычислительных кластеров и вычислительных GRID и требуют специальных инструментов организации вычислений. В статье представлен механизм динамического прогнозирования времени завершения вычислительного эксперимента в Desktop Grid. Мы предлагаем статистический подход, основанный на линейной регрессионной модели с вычислением доверительного интервала, учетом накопления статистической ошибки и соответствующим – при необходимости – изменением прогноза. На основе предложенного подхода разработан алгоритм прогнозирования и программный модуль для Desktop Grid на базе BOINC. Мы представляем результаты экспериментов, основанные на данных проекта добровольных вычислений RakeSearch.

**Ключевые слова:** Desktop Grid; BOINC; время выполнения; временной ряд; точечный прогноз; доверительный интервал; статистическая ошибка

**Для цитирования:** Ивашко Е.Е., Литовченко В.С. Динамическое построение прогноза времени завершения вычислительного эксперимента в Desktop Grid. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 183-190. DOI: 10.15514/ISPRAS-2019-31(5)-14

**Благодарности.** Работа выполнена при поддержке грантов РФФИ: №18-37-00094 мол_а, РФФИ №18-07-00628 А

# Dynamic forecasting of the completion time of a computational experiment in a Desktop Grid

[1,2] *E.E. Ivashko ORCID: 0000-0001-9194-3976 <ivashko@krc.karelia.ru>*
[2] *V.S. Litovchenko ORCID: 0000-0003-1104-0502 <va.lentina97@yandex.ru>*

[1] *Institute of Applied Mathematical Research, Karelian Research Centre of RAS,*
*11, Pushkinskaya st., Petrozavodsk, 185910, Russia*
[2] *Petrozavodsk State University,*
*35, Lenin Avenue, Petrozavodsk, 185910, Russia*

**Abstract.** Desktop Grid is a powerful tool to perform high-throughput computing. Desktop Grid is a form of distributed high-throughput computing system, which uses idle time of non-dedicated geographically distributed computing nodes connected over low-speed network. It has significant differences from computing clusters and computational GRIDs, and needs special operation tackle. In this paper, we present a mechanism for dynamic forecasting of the completion time of a computational experiment in a Desktop Grid. We propose a statistical approach based on the linear regression model with the calculation of a confidence interval, taking into account the accumulation of statistical error and, if needed, changing the forecast. The developed approach is used to implement a forecasting algorithm and software module for a Desktop Grid. We present experimental results based on data from the RakeSearch volunteer computing project.

**Keywords:** Desktop Grid; BOINC; tasksbag runtime estimation; completion time; point prediction; confidence interval; standard error of estimation

**For citation:** Ivashko E.E., Litovchenko V.S. Dynamic forecasting of the completion time of a computational experiment in a Desktop Grid. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019, pp. 183-190 (in Russian). DOI: 10.15514/ISPRAS-2019-31(5)-14

**Acknowledgements.** This work was supported by RFBR grants: №18-37-00094, №18-07-00628 A.

## *1. Введение*

В различных областях науки и техники существуют проблемы, требующие больших вычислительных ресурсов для нахождения решения. Системы типа Desktop Grid хорошо подходят для выполнения научных высокопроизводительных расчетов [1]. Такие системы имеют вычислительный потенциал, превышающий 1 ExaFLOPS [2]. Desktop Grid – это форма распределенной высокопроизводительной вычислительной системы, которая использует время простоя географически распределенных вычислительных узлов, объединенных низкоскоростной сетью. Вычислительные системы такого типа имеют ряд преимуществ: высокая масштабируемость, надежность, низкая стоимость создания и сопровождения вычислительной системы и др.

Благодаря своим преимуществам, системы типа Desktop Grid являются популярными инструментами для решения вычислительно емких научных задач типа «bag of tasks». Вычислительный процесс в Desktop Grid выглядит следующим образом. Вычислительно сложная задача делится на большое количество небольших подзадач; сервер Desktop Grid отправляет подзадачи на вычислительные узлы (клиенты); когда результат вычислений готов, клиент отправляет его обратно на сервер и запрашивает новое подзадание. После того, как на сервере собраны все результаты, они объединяются в единое решение исходной задачи [1]. Подробное описание вычислительного процесса в Desktop Grid на основе BOINC приведено в [3].

Зачастую в рамках вычислительных проектов проводятся отдельные вычислительные эксперименты (или «кампании» (campaign) в терминах работы [4]). Вычислительный эксперимент представляет собой набор заданий, для которого анализ результатов может быть начат только после завершения всех задач вычислительного эксперимента.

Соответственно, возникает проблема оценки времени выполнения набора заданий; для исследователя важно знать, когда он может начать обработку результатов вычислительного эксперимента. Существует ряд особенностей, которые осложняют оценку времени выполнения набора заданий в Desktop Grid:

- высокая аппаратная и программная гетерогенность;
- низкая надежность вычислительных узлов;
- неопределенность времени обработки.

Оценка времени выполнения является важной проблемой для вычислительных проектов, основанных на системах типа Desktop Grid. В данной статье мы представляем процедуру динамической оценки времени выполнения набора заданий. Работа расширяет результаты, представленные в статье [5] ранее. В данной работе мы предлагаем статистический подход, основанный на линейной регрессионной модели, расчете доверительного интервала и динамическом пересчете прогноза с учетом накопленной статистической ошибки.

## *2. Обзор работ*

Ряд научных работ посвящен оценке времени выполнения набора заданий. Авторы статьи [6] рассматривают и сравнивают наиболее популярные промежуточные программные обеспечения для реализации Desktop Grid. Программная платформа BOINC является самым популярным программным обеспечением организации Desktop Grid для выполнения добровольных вычислений [7]; она имеет встроенный механизм оценки времени выполнения одной задачи. При этом платформа BOINC проста в развертывании, использовании и управлении [8]. Однако ее нельзя использовать для оценки времени выполнения набора заданий.

Можно отметить, что в случае постоянного значения доступных вычислительных ресурсов и идентичных (с вычислительной точки зрения) задач время выполнения измеряется по формуле: $Runtime(n) = n \cdot w/R$, где $Runtime(n)$ – время, требубщееся для выполнения $n$ задач, $w$ – время завершения одной задачи, $R$ – доступные вычислительные ресурсы. Такая формула в общем случае не может применяться для реальных проектов Desktop Grid. Этот простой подход можно немного улучшить. Действительно, в случае колебаний доступных ресурсов можно использовать усреднение; в случае линейного увеличения / уменьшения ресурсов / сложности задачи следует измерять тренд. Мы используем статистический подход, который больше подходит для проектов Desktop Grid и охватывает все эти случаи.

Авторы статьи [9] описывают подход к планированию тысяч заданий с различными требованиями к неоднородным ресурсам Grid, которые используют компьютеры добровольцев с программным обеспечением BOINC. Ключевой компонент этой системы обеспечивает априорную оценку времени выполнения с использованием машинного обучения со случайными лесами.

В статье [4] автор рассматривает проблему сокращения времени выполнения набора заданий («кампании»). Важной частью решения является оценка «времени завершения компании» $\tau_c$, которое зависит от доступных вычислительных ресурсов $R_A$ для соответствующего приложения $A$, начинающего компанию $n_A$, количества незавершенных задач $W_c$ и текущего времени $t$: $\tau_c = t + W_c \cdot n_A/R_A$. Автор использует точечный прогноз, который обновляется каждый раз, когда изменяется количество доступных ресурсов. Этот подход имеет явные недостатки: во-первых, в реальном проекте Desktop Grid количество доступных ресурсов неизвестно из-за неизвестного количества работающих вычислительных узлов; во-вторых, этот подход не учитывает изменчивую производительность; наконец, предполагается, что в любое время число незавершенных задач известно, а это не так. Кроме того, этот подход показывает гораздо

лучшие результаты, чем обычный механизм самой популярной программной платформы Desktop Grid BOINC.

В статьях [10] и [11] представлен механизм планирования задач, который основан на прогнозировании времени выполнения для задач подбора параметра. Используя этот подход, можно сопоставить прогнозируемые окна доступности ресурсов с прогнозируемым временем выполнения отдельных задач. Это позволяет уменьшить количество задач, отмененных из-за нехватки вычислительного времени на узле. Подход реализован в программном обеспечении GIPSy (Grid Information Prediction System). Авторы также предлагают несколько прогнозных математических моделей.

Авторы статьи [12] пытаются оценить верхние временные границы задержек, которые являются частью времени жизни задачи (задержки распределения, выполнения и валидации). В статье авторы оценивают точность нескольких вероятностных методов. Более точное прогнозирование времени ожидания позволяет более точно прогнозировать время выполнения набора заданий, обеспечивает более эффективное использование ресурсов, более высокую пропускную способность проекта и более низкую задержку задания в проектах Desktop Grid.

В статье [13] предложен подход динамической репликации для сокращения времени, необходимого для выполнения набора заданий. Предложены математическая модель и стратегия динамической репликации на стадии хвоста вычислений.

В работе [14] авторы формулируют аналитическую модель, которая позволяет сравнивать различные политики распределения ресурсов. В частности, авторы изучают политику распределения ресурсов, которая направлена на минимизацию среднего времени завершения работы; показано, что предлагаемая политика может сократить среднее время завершения на 50% от необходимого для политик равномерного или линейного распределения. В статье [15] Desktop Grid динамически дополняется облаком Инфраструктура как услуга (IaaS) для сокращения среднего времени выполнения задач.

Часть главы [16] посвящена обзору и описанию нескольких инструментов моделирования Grid и Desktop Grid: SimBA, SimBOINC и EmBOINC software. В работе [17] также рассматривается моделирование различных вычислительных систем с фокусировкой на симуляторе SimGrid. Работы [18] и [19] посвящены эмуляции Desktop Grid на EmBOINC, также авторы обращают внимание на несколько средств моделирования. В работе [20] описан симулятор BOINC SimBA.

Эта статья расширяет работу [5], в которой модель Хольта предлагается в качестве модели прогнозирования. Мы провели численные эксперименты на базе статистики работы проекта RakeSearch. Результаты экспериментов показывают хорошее приближение точечного прогноза к реальному значению. Однако на практике доверительный интервал важнее точечного прогноза. Эксперименты показывают хорошее покрытие реальных данных доверительными интервалами. Это показывает, что, несмотря на высокую неоднородность и большое количество вычислительных узлов, а также различную сложность задач, можно получить полезную оценку времени завершения. Существует ряд нюансов, которые необходимо учитывать при реализации описанного подхода. Во-первых, точность аппроксимации моделью Хольта зависит от коэффициентов сглаживания ряда и тренда. Во-вторых, при прогнозировании на большое число шагов вперед целесообразно выполнять масштабирование временного ряда. В-третьих, для расчета доверительного интервала берутся $k$ последних точек. В-четвертых, на практике необходим динамический пересчет прогноза при поступлении новых результатов. В этой статье мы используем модель линейной регрессии (которая на практике лучше подходит) и предлагаем метод динамического прогнозирования, основанный на накоплении статистической ошибки.

## *3. Динамическое построение прогноза*

Мы предлагаем статистический подход, основанный на следующих предположениях. Во-первых, существует функциональная зависимость между прошлыми и будущими значениями процесса. Во-вторых, эта зависимость имеет кусочно-линейный вид с восходящим трендом. С точки зрения Desktop Grids эти предположения означают следующее. Наблюдаемый процесс описывает моменты времени получения новых результатов. Таким образом, он является строго возрастающим (два результата не могут быть получены одновременно). Угол линии тренда описывает производительность Desktop Grid. Производительность может варьироваться, а тренд меняться соответственно. Мы предполагаем, что изменение производительности является линейным, потому что нелинейные изменения обычно связаны с началом проекта, вычислительными соревнованиями или другими нерегулярными периодами проекта. Такие нелинейные эффекты ограничены переходным периодом и быстро исчезают.

Построение оценки времени выполнения состоит из трех этапов: сначала строится точечная оценка времени завершения и соответствующий доверительный интервал, затем в ходе выполнения расчетов отслеживается накопление статистической ошибки, наконец, при достижении статистической ошибкой некоторого заданного порога выполняется пересчет прогноза с обновлением точечной оценки времени завершения и доверительного интервала.

Рассмотрим эти три этапа более подробно. В соответствии с нашими предположениями, прогнозируемый ряд имеет кусочно-линейный вид. Соответственно, линейная регрессия хорошо подходит для прогнозирования линейных частей. Линейная регрессия определяется формулой [21]: $y_i = a \cdot x_i + b + \varepsilon_i$. Здесь коэффициенты $a, b$ находятся по методу наименьших квадратов [22].

Для предоставления прогноза ученому, выполняющему вычислительный эксперимент, мы рассчитываем доверительный интервал. Доверительный интервал – это интервал, в котором с определенной долей вероятности лежит истинное значение статистической характеристики. Более подробное описание математических формул, связанных с линейной регрессией, методом наименьших квадратов и доверительным интервалом, можно найти, например, в [5].

Второй этап – это накопление статистической ошибки. Статистическая ошибка представляет собой стандартное отклонение наблюдаемых значений от прогнозируемых и рассчитывается по формуле [23]: $\varepsilon_i = \sum_{i=1}^{n} |y_i - y_i^*| / n$, где $y_i$ – расчетные значения. При превышении накопленной ошибкой определенного уровня, считается, что прогноз не соответствует реальному процессу. Применительно к нашей задаче это означает, что в процессе расчетов сменился тренд и производительность Desktop Grid изменилась. Поэтому прогноз должен быть пересчитан. Пример прогнозирования, накопления статистической ошибки и смены прогноза представлена на рис. 1.

По оси X откладывалась накопленная работа, по оси Y время. Применимость подхода оценивалась на основе численных экспериментов, проводимых с использованием истории работы проекта добровольных вычислений RakeSearch [24]. Для оценки времени завершения брались наборы заданий случайной продолжительности (от 100 до 1000 подзаданий) со случайным временем упреждения (от 10 до 50 подзаданий). На первом шаге мы рассчитали коэффициенты a и b. По подобранному уравнению регрессии мы находили прогнозные значения, строили прогноз и считали стандартную ошибку. Как только ошибка достигала 8% мы перестраивали прогноз.

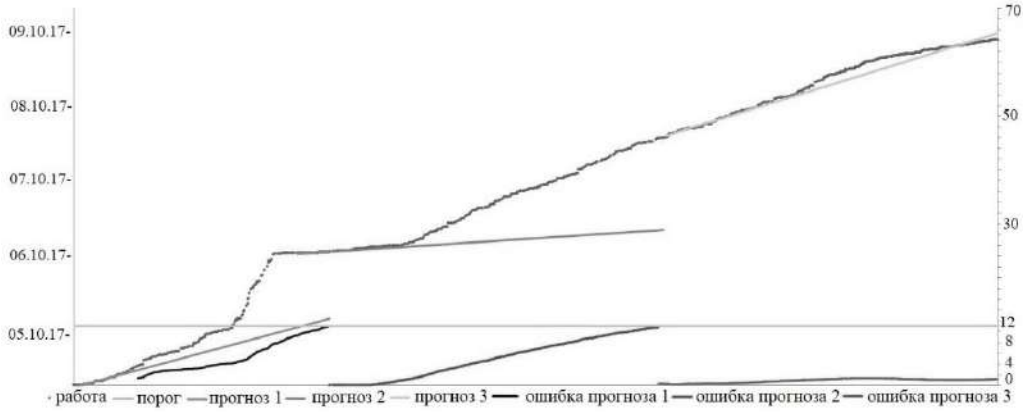


*Рис. 1. Прогнозирование и статистическая ошибка*
*Fig. 1. Forecasting and statistical error*

### *4. Заключение*

Вычисления в рамках Desktop Grid активно используются в научной сфере. Так как вычислительные проекты Desktop Grid часто основаны на вычислительных экспериментах, оценка времени выполнения набора заданий является важной с практической точки зрения задачей. Работа посвящена оценке времени выполнения набора заданий. Мы представили статистический подход к оценке времени выполнения набора заданий. В качестве статистической модели была выбрана модель линейной регрессии. Исходя из наших предположений, эта модель подходит для прогнозирования линейных частей.

Мы предложили процедуру прогнозирования, состоящую из трех этапов: построение точечной оценки времени завершения и соответствующего доверительного интервала, отслеживание накопления статистических ошибок и пересчет прогноза при необходимости. Мы провели численные эксперименты, используя историю проекта добровольных вычислений RakeSearch. Разработанная процедура прогнозирования позволила с большей точностью прогнозировать значения на более длительный период шагов вперед.

## Список литературы / References

[1]. Ивашко Е.Е., Никитина Н.Н. Использование BOINC-грид в вычислительноемких научных исследованиях. Вестник Новосибирского государственного университета. Серия: Информационные технологии, том 11, вып. 1, 2013 г., стр. 53-57 / E.E. Ivashko, N.N. Nikitina. Employment of boinc-grid in computationally intensive scientific research. Vestnik NSU. Series: Information Technologies, vol. 11, issue 1, pp. 53-57 (in Russian).

[2]. Wu W., G. Chen, W. Kan, D. Anderson, F. Grey. Harness public computing resources for protein structure prediction computing. In Proc. of the International Symposium on Grids and Clouds (ISGC), 2013.

[3]. Anderson D. P., Christensen C., Allen B. Designing a Runtime System for Volunteer Computing. In Proc. of the 2006 ACM/IEEE Conference on Supercomputing, 2006, p. 126.

[4]. Amstel D. van. Scheduling for Volunteer Computing on the BOINC server infrastructure. PhD Thesis, 2012.

[5]. Ivashko E., Litovchenko V. Batch of Tasks Completion Time Estimation in a Desktop Grid. Communications in Computer and Information Science, vol. 965, 2019, pp. 500–510.

[6]. Khan M.Kh., Mahmood T., Hyder S.I. Scheduling in Desktop Grid Systems: Theoretical Evaluation of Policies and Frameworks. International Journal of Advanced Computer Science and Applications, vol. 8, issue 1, 2017, pp. 119–127.

[7]. Ивашко Е.Е. Desktop Grid корпоративного уровня. Программные системы: теория и приложения, том 5, № 1, 2014, стр. 183-190 / Ivashko E.E. Enterprise Desktop Grids. Program Systems: Theory and Applications, vol. 5, № 1, pp. 183-191 (in Russian).
[8]. Anderson D.P. A system for public-resource computing and storage. In Proc. of the Fifth IEEE/ACM International Workshop on Grid Computing, 2004, pp. 4–10.
[9]. Bazinet A. L., Cummings M. P. Computing the Tree of Life: Leveraging the Power of Desktop and Service Grids. In Proc. of the International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, 2011, pp. 1896–1902.
[10]. Peter Hellinckx, Sam Verboven, Frans Arickx, Jan Broeckhove. Predicting Parameter Sweep Jobs: From Simulation to Grid Implementation. In Proc. of the 2009 International Conference on Complex, Intelligent and Software Intensive Systems, 2009, pp. 402–408.
[11]. Hellinckx P., Verboven S., Arickx F., Broeckhove J. Runtime prediction based Grid scheduling of parameter sweep jobs. In Proc. of the IEEE Asia-Pacific Services Computing Conference, 2008, pp. 33–38.
[12]. Estrada, T., Taufer M. and Reed K. Modeling Job Lifespan Delays in Volunteer Computing Projects. In Proc. of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009, pp. 331-338.
[13]. Kolokoltsev Y., Ivashko E., Gershenson C. Improving “tail” computations in a BOINC-based Desktop Grid. Open Engineering, vol. 7, issue 1, 2017, pp. 119–127.
[14]. Villela D. Minimizing the Average Completion Time for Concurrent Grid Applications / Journal of Grid Computing, 2010, vol. 8, issue 1, pp. 47–59.
[15]. Reynolds C. J., Winter S., Terstyanszky G.Z., Kiss T., Greenwell P., Acs S., Kacsuk P. Scientific Workflow Makespan Reduction through Cloud Augmented Desktop Grids. In Proc. of the IEEE Third International Conference on Cloud Computing Technology and Science, 2011, pp. 18–23.
[16]. Estrada T., Taufer M. Challenges in Designing Scheduling Policies in Volunteer Computing. In Desktop Grid Computing, Chapman & Hall/CRC, 2012, pp. 167–190.
[17]. Beaumont O., Bobelin L., Casanova H. et al. Towards Scalable, Accurate, and Usable Simulations of Distributed Applications and Systems. Institut National de Recherche en Informatique et en Automatique, Research Report RR-7761, 2011.
[18]. Estrada T., Taufer M., Anderson D.P. Performance Prediction and Analysis of BOINC Projects: An Empirical Study with EmBOINC. Journal of Grid Computing, no. 7, 2009, pp. 537-554.
[19]. Estrada T., Taufer M., Reed K., Anderson D.P. EmBOINC: An emulator for performance analysis of BOINC projects. In Proc. of the International Symposium on Parallel & Distributed Processing, 2009, pp. 1–8.
[20]. Taufer M., Kerstens A., Estrada T., Flores D.A., Teller P.J. SimBA: A Discrete Event Simulator for Performance Prediction of Volunteer Computing Projects. In Proc. of the21st International Workshop on Principles of Advanced and Distributed Simulation, 2007, pp. 189–197.
[21]. Чучуева И.А. Модель прогнозирования временных рядов по выборке максимального правдоподобия. Диссертация на соискание ученой степени кандидата технических наук. МГТУ, 2012 / Chuchueva I.A. Time Series Prediction Model for Maximum Credibility Sampling. The dissertation for the degree of candidate of technical sciences. MSTU, 2012 (in Russian).
[22]. Четыркин Е М. Статистические методы прогнозирования. М., Статистика, 1977, 200 стр. / Chetyrkin E. M. Statistical methods of forecasting. M., Statistics, 1977, 200 p. (in Russian).
[23]. Мамаева З.М. Введение в эконометрику. Нижегородский госуниверситет, 2010, 72 стр. / Mamaeva Z.M. Introduction to Econometrics. Nizhny Novgorod State University, 2010, 72 p. (in Russian).
[24]. Manzyuk M., Nikitina N., Vatutin E. Employment of Distributed Computing to Search and Explore Orthogonal Diagonal Latin Squares of Rank 9. In Proc. of the XI All-Russian research and practice conference "Digital techologies in education, science, society", 2017, pp. 97-100.

## Информация об авторах / Information about authors

Евгений Евгеньевич ИВАШКО – кандидат физико-математических наук, руководитель ЦКП КарНЦ РАН «Центр высокопроизводительной обработки данных», доцент ПетрГУ. Сфера научных интересов: высокопроизводительные и распределенные вычисления, Desktop Grid, математическое моделирование, разработка информационных и информационно-аналитических систем.

Evgeny Evgenievich IVASHKO – PhD in physics and mathematics, head of High-performance computing centre of KRC of RAS, assistant professor of Petrozavodsk State University. Research interests: high-throughput and distributed computing, Desktop Grid, mathematical modelling, information and analytics platform development.

Валентина Степановна ЛИТОВЧЕНКО является студентом направления "Математическое моделирование и информационно-коммуникационные технологии" Петрозаводского государственного университета. Сфера научных интересов: высокопроизводительные и распределенные вычисления, Desktop Grid, математическое моделирование.

Valentina Stepanovna LITOVCHENKO is a student in the field of "Mathematical Modeling and Information and Communication Technologies" at Petrozavodsk State University. Research interests: high-performance and distributed computing, Desktop Grid, mathematical modeling.

**DOI: 10.15514/ISPRAS-2019-31(5)-15**



# Machine Learning Use Cases in Cybersecurity

[1] *S.M. Avdoshin, ORCID: 0000-0001-8473-8077 <savdoshin@hse.ru>*
[2] *A.V. Lazarenko, ORCID: 0000-0001-5812-0134 <lazarenko@group-ib.com>*
[2] *N.I. Chichileva, ORCID: 0000-0002-3012-8043 <chichileva@group-ib.com>*
[2] *P. A. Naumov, ORCID: 0000-0002-9323-9074 <naumov@group-ib.com>*
[3] *P. G. Klyucharev, ORCID: 0000-0001-9536-8083<pk.iu8@yandex.ru>*

[1] *National Research University Higher School of Economics, 20, Myasnitskaya st., Moscow, 101000 Russia*
[2] *Group-IB, 1, Sharikopodshipnikovskaya Ulitsa, Moscow, 115080 Russia*
[3] *Bauman Moscow State Technical University, 5/1, 2nd Baumanskaya Ulitsa, Moscow, 105005 Russia*

**Abstract.** The problem regarding the use of machine learning in cybersecurity is difficult to solve because the advances in the field offer many opportunities that it is challenging to find exceptional and beneficial use cases for implementation and decision making. Moreover, such technologies can be used by intruders to attack computer systems. The goal of this paper to explore machine learning usage in cybersecurity and cyberattack and provide a model of machine learning-powered attack.

**Keywords:** cyberattack; cybersecurity; deep learning; machine learning; machine learning-powered cyberattack

**Для цитирования:** Avdoshin S.M., Lazarenko A.B., Chichileva N.И., Naumov P.A., Klyucharev P.Г. Machine Learning Use Cases in Cybersecurity. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019, pp. 191-202. DOI: 10.15514/ISPRAS-2019-31(5)-15

## Примеры использования машинного обучения в кибербезопасности

[1] *С.М. Авдошин, ORCID: 0000-0001-8473-8077 <savdoshin@hse.ru>*
[2] *А.В. Лазаренко, ORCID: 0000-0001-5812-0134 <lazarenko@group-ib.com>*
[2] *Н.И. Чичилева, ORCID: 0000-0002-3012-8043 <chichileva@group-ib.com>*
[2] *П.А. Наумов, ORCID: 0000-0002-9323-9074 <naumov@group-ib.com>*
[3] *П.Г. Ключарев, ORCID: 0000-0001-9536-8083<pk.iu8@yandex.ru>*

[1] *НИУ Высшая школа экономики, 101978, Россия, г. Москва, ул. Мясницкая, д. 20*
[2] *Group-IB, 115080, Россия, г. Москва, ул. Шарикоподшипниковская, д. 1*
[3] *Московский государственный технический университет им. Н.Э.Баумана, 105005, г. Москва, 2-я Бауманская ул., д. 5, стр. 1*

**Аннотация.** Проблему использования машинного обучения в кибербезопасности трудно решить, поскольку достижения в этой области открывают так много возможностей, что сложно найти действительно хорошие варианты решения реализации и принятия решений. Более того эти технологии также могут использоваться злоумышленниками для кибератаки. Цель этой статьи –

сделать обзор актуальных технологий в кибербезопасности и кибератаках, использующих машинное обучение, и представить модель атаки на основе машинного обучения.

**Ключевые слова:** кибератака; кибербезопасность; глубокое обучение; машинное обучение; кибератака с машинным обучением

**Для цитирования:** Авдошин С.М., Лазаренко А.В., Чичилева Н.И., Наумов П.А., Ключарев П.Г. Примеры использования машинного обучения в кибербезопасности. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 191-202 (на английском языке). DOI: 10.15514/ISPRAS-2019-31(5)-15

## 1. Introduction

Cybersecurity is gaining more and more attention each Cybersecurity is gaining more and more attention each year. The number of cyberattacks has significantly increased since 2009 due to the digitalization of everything in the modern world. According to the Gartner Hype Cycle [1], machine learning (ML) is of great interest in the world of technology. ML is concerned with intelligent behaviour in a system, including perception, reasoning, learning, communication and acting in a complex environment [2]. Such widespread interest in ML is due to two critical factors: First, it can automate processes that previously required human participation, for example, control of robotic mechanisms in production (i.e. ML assumes human responsibilities). Second, it can quickly process and analyze huge amounts of information and calculate options using many variables. In these areas, ML provides qualitatively better results compared to humans.

ML has much to offer cybersecurity. Current implementations are widely used in IDS systems, sandbox systems and many different areas of cybersecurity – from threat intelligence data collection to advanced automated digital forensics. In fact, 71% of US businesses plan to use ML in their cybersecurity tools in 2019 [3] as over one-third (36%) [3] of organizations experienced damaging cyberattacks in 2018. The majority (83%) [3] confides that cybercriminals use ML to attack organizations. The problem of ML use in cybersecurity is difficult to solve because the advances in the field offer so many opportunities that it is hard to find good and beneficial use cases for implementation and decision making. Moreover, it is difficult to determine how secure a security system is, which is used in production, and how to protect the organization from cyberattacks conducted through ML. The main goal of the current work explore ML usage in cybersecurity and research use cases related to the adversary's use of ML in cyberattacks.

## 2. Basic definitions

ML is the process by which machines learn from given data, building the logic and predicting output for a given input [4]. ML has three sub-categories: supervised learning, unsupervised learning and reinforcement learning [5]. Supervised learning uses a dataset labelled with the correct answers for study. Such labels identify the characteristics of each dataset. Once the model is trained, it can start predicting or deciding on new data that is given to it. In unsupervised learning, there is no need for such a marked set of data. Once the model is given the dataset, it automatically finds patterns and relationships by creating clusters in it. However, such type of learning cannot predict anything. When new data is added, the model assigns them to one of the existing clusters or creates a new one. Reinforcement learning is the ability of a system to interact with the environment and identify the best outcome. The system is either rewarded or penalized with a point for a correct or a wrong answer, and based on positive reward points gained, the model trains itself. Similarly, once trained, it prepares to predict new data presented to it.

Deep learning (DL) is a class of ML algorithms [6] that uses multiple layers to progressively extract higher-level features from the raw input. The main differences between ML and DL are as follows: ML algorithms almost always require structured data, whereas DL networks rely on layers of the Artificial Neural Networks (ANNs). Often in ML, human intervention is necessary

to produce further outputs with more sets of data, while in DL, this is not necessary. One of the central concepts in DL is ANNs. The ANN is a model that is built on the principle of organization and the functioning of the human brain (i.e. networks of nerve cells in a living organism). In other words, a neural network algorithm tries to create a function to map one's input to one's desired output. Neural networks (NNs) are typically organized in layers (fig. 1). Layers consist of a number of interconnected 'nodes' that contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer' where the answer is the output.

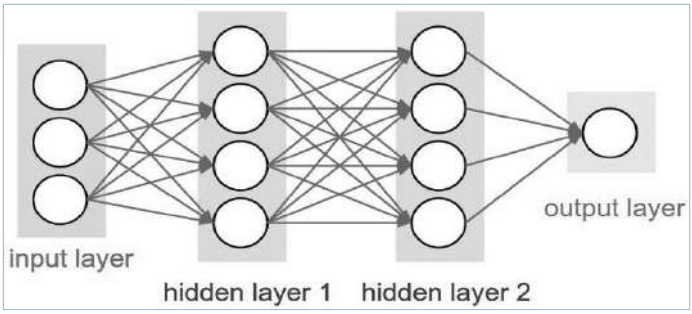


*Fig. 1. Neural networks*

For example, in image processing, lower layers may identify edges, while higher layers may identify concepts relevant to a human, such as digits, letters or faces. If NNs have more than two hidden layers, they are called deep neural networks (DNNs) [7]. DNN is used for image recognition, speech recognition and other applications. Moreover, technologies have been created to generate new photographs that look at least superficially authentic to human observers through many realistic characteristics. For example, there is a known attempt to synthesize photographs of cats that has misled an expert to think they are real ones [8]. This is an example of the technology called generative adversarial network (GAN), an ML algorithm of unsupervised learning built on a combination of two NNs: one network G (generator) generates new examples and one network D (discriminator) tries to classify examples as either real or fake (generated) [9].

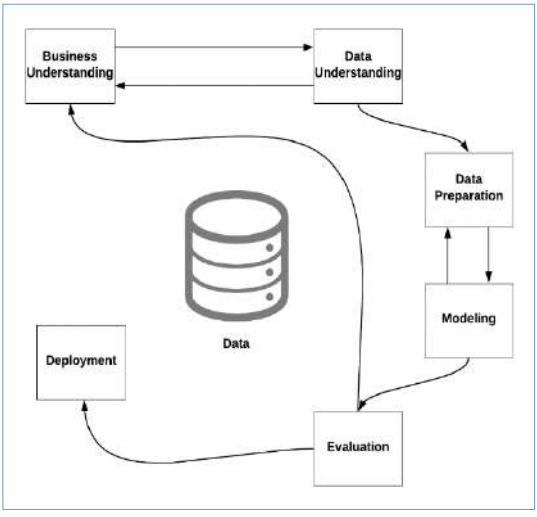


*Fig. 2. CRISP-DM process of data mining*

One of the processes that is inextricably linked with ML and DL is data mining. Using data mining in large datasets can identify new patterns by utilizing statistics and database systems methods [10]. The Cross-Industry Standard Process for Data Mining (CRISP-DM) describes the cross-industry process for data mining [11]. CRISP-DM breaks the process into six main phases: business understanding, data understanding, data preparation, modelling, evaluation and deployment (fig. 2). The first two phases are connected to each other. Their main aim is to determine the goals of the project, set the task for ML and collect data. These aims can be adjusted based on the data. The next phase refers to the process of working with data: cleaning the data, combining the data, if necessary, and formatting the data.

In the modelling phase, various modelling techniques are applied to the data. Models are built, and their parameters are adjusted to optimal values. Because of special data requirements in different models, we can return to the Data Preparation phase. In the evaluation phase, the model has already been built, and quantitative assessments of its quality have been obtained. Before implementing this model, we need to make sure that we have achieved all business goals. Depending on the requirements, the deployment phase may be simple (e.g. preparation of the final report) or complex (e.g. automation of the data analysis process to solve business problems).

## *3. Using ML for protection*

The scope of ML usage in cybersecurity is huge, starting with identifying anomalies and suspicious or unusual behaviours and ending with detecting zero-day vulnerabilities and patching known ones. Dilek et al. [12] presented the most comprehensive review of applications of ML techniques.

Reathi and Malathi [13] presented a set of ML algorithms trained on the NSL-KDD intrusion detection dataset for misuse detection. Meanwhile, Buczak et al. [14] focused on network intrusion detection using ML.

Melicher et al. [15] proposed using NNs to check password guessing resistance. They compressed the model to hundreds of kilobytes and developed a client-side JavaScript tool. The similar experiment was conducted by Ciaramella et al. [16]. To proactively check the strength of passwords, they use NNs, such as Multilayer Perceptron (MLP) and Single Layer Perceptrons (SLPs). Notably, MLPs provide better results than SLPs when testing datasets. Moreover, the number of layers equal to 10, and thus obtains better result.

User and entity behaviour analytics (UEBA) use ML capabilities to analyze behaviour logs and network traffic in real-time and respond appropriately in the event of an attack [17]. This process is done by getting the user to log in again, blocking an attack or assessing risk levels and alerting the company's information security officers so that they can take necessary action.

Most of the ML and DL methods, such as ensemble learning, clustering, and decision tree, [18] are used to detect misuse, anomaly and hybrid cyber intrusion.

As mentioned in the Eugene Kaspersky Official Blog [19], Kaspersky detects 99% of cyber threats using ML technology. The time interval between the disclosure of suspicious behaviour on the protected device and the release of the corresponding new 'tablet' lasts an average of 10 minutes.

DARPA collaborated with BAE Systems to develop a system that allow us to configure sensors and apply protective measures 'at machine speed'. This initiative called the CHASE program, which stands for Cyber Hunting at Scale, seeks to develop automated tools to detect and characterize novel attack vectors, collect the right contextual data, and disseminate protective measures both within and across enterprises [20].

Cyberattacks performed by hacktivists relate to a common opinion about high-profile news. Information gathered from social media can help predict such incidents using NLP and ML techniques [21].

Moreover, we can use ML to identify the author of the program. Rachel Greenstadt and Aylin Caliskan developed a system that can 'deanonymize' programmers [22] by analyzing source code or compiled binary files [23]. Identifying the developer of malware is now much easier.

Another way to monitor systems and networks for malicious activity or policy violation is through the intrusion detection system (IDS). Intrusion prevention system (IPS) is a system connected with IDS; these systems perform intrusion detection and stop the detected incidents. Both systems use supervised and unsupervised ML techniques to detecting point anomaly, contextual anomaly, and collective anomaly [24].

The main task of firewalls [25] is to ensure a network security system that monitors and controls incoming and outgoing network traffic. Firewalls allow or block traffic by comparing its characteristics with predefined patterns (i.e. firewall rules). In their paper, Ucar and Ozhan [26] presented the result of the automatic detection of anomalies in firewall rule repository based on ML and high-performance computing methods, such as Naive Bayes, kNN, Decision Table and HyperPipes. All six firewall rules from the given 93 rules were detected by the system and verified by the experts as an anomaly. Firewalls filter the content between servers, and there is also a solution specifically meant for the content of web applications. Web application firewall (WAF) is deployed in front of web applications; it analyzes bi-directional web-based (HTTP) traffic and detects and blocks anything malicious [27]. WAF prevents vulnerabilities in web applications from being exploited by outside threats. To implement such functionality in WAF, developers use regular expressions, tokens, behavioural analysis, reputation analysis and ML technologies [27].

Among ML methods, special predictive ones can also be used for data loss/leak prevention (DLP) to reduce the risk for breaches or leaks [28]. DLP software solutions allows us to set business rules that classify confidential and sensitive information so that they cannot be disclosed maliciously or accidentally by unauthorized end users. This process can be done by using supervised learning algorithms and two types of examples: positive examples (i.e. content that needs to be protected) and counterexamples (i.e. documents that are similar to the positive set but should not be protected).

## *4. Using ML in cyberattacks*

This section describes how cyberattack can succeed using ML. Automated vulnerability scanning is one of the most obvious and common tasks in a cyberattack. For example, CSRF is found in only 5% of applications, as reported in the 2017 OWASP Top 10, because most frameworks include CSRF defences [29]. Accordingly, Calzavara et al. presented Mitch [30], the first ML-based tool for the black-box detection of CSRF, which allows the identification of 35 new CSRF vulnerabilities on 20 websites from the Alexa Top 10,000 websites and three previously undetected CSRF vulnerabilities on production software already analyzed with the state-of-the-art tool Deemon [31]. Mitch is a binary classifier, labelling sensitive or insensitive requests using a random forest algorithm on a 49-dimensional feature space. Compared to the heuristic classifiers BEAP [32] and CsFire [33], Mitch shows the best F1-score and precision (Table 1).

Marketers use ML methods for profiling. Trustwave released an open source intelligence tool that uses face recognition to automatically track subjects across social media networks [34]. Facial recognition aids this process by removing false positives in the search results, making data review faster for a human operator.

*Table 1. Validity measures for the tested classifiers (BEAP, CsFire, Mitch)*

| Classifier | Precision | Recall | F1 |
|---|---|---|---|
| BEAP | 0.30 | 0.89 | 0.45 |

| | | | |
|---|---|---|---|
| CsFire | 0.20 | 0.97 | 0.33 |
| Mitch | 0.78 | 0.67 | 0.72 |

Using collected data about the target, an attacker can hook a victim with specially created fake news. ML tools can help identify fake news, but to do so, researchers confirm that the best way is for that ML to learn to create fake news itself [35]. As such, they created a model for controllable text generation called Grover. In the research process, four classes of articles were used: human news, machine news, human propaganda and machine propaganda. Workers on Amazon Mechanical Turk rated each article, including overall trustworthiness. In the case of propaganda, the score increased from 2.19 (out of 3) on articles created manually to 2.42 on articles created by a machine.

SNAP_R was introduced at DEFCON 24. SNAP_R is the world's first automated end-to-end spear-phishing campaign generator for Twitter [36]. While previous tools were based on models with Markov chains, SNAP_R is based on a recurrent NN with LSTM architecture. Using Twitter as an environment offers some advantages for automatically generating text. For example, limiting the length of a post decreases the probability of grammatical errors. Moreover, Twitter links are often shortened, which allows masking of malicious domains. This, in turn, significantly increased the success rate from 5–14% on Markov chain-based tools [37, 38] to 30–66%, which is comparable to the 45% rate for manual spear-phishing [39].

In most cases, attackers do not know the malware detection algorithm but can figure out features it uses through carefully designed test cases in the black-box algorithm. MalGAN is a generative adversarial network-based algorithm that generates adversarial malware examples that are able to bypass black-box ML-based detection models. It can decrease the detection rate to nearly zero and make it hard for the retraining-based defensive method against adversarial examples to work [40]. The architecture of MalGAN is shown in fig. 3 [40].

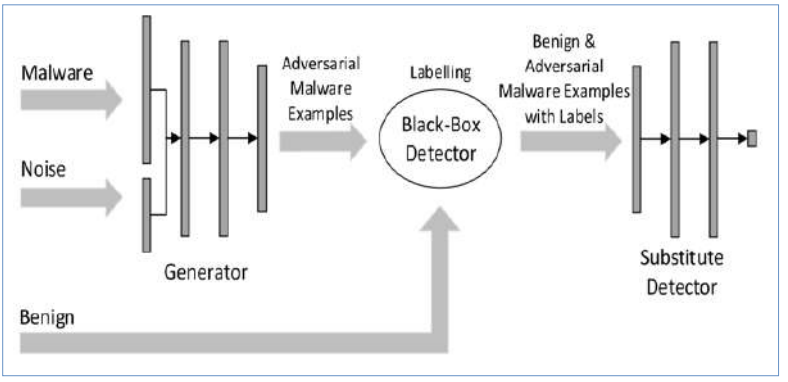


*Figure 3. Architecture of MalGAN*

The generator takes the malware feature vector and the noise vector to transform the former into its adversarial version. Substitute detector is used to fit the black-box detector and provide gradient information to train the generator. Both nets are represented as multi-layer feed-forward ANNs. Adversarial examples tested against the black-box detector according to different ML methods trained on 160-dimensional binary feature vectors representing system API calls include random forest, logistic regression, decision trees, support vector machines, and multi-layer perceptron as well as a voting-based ensemble of these algorithms. All these classifiers detect over 90% of original samples, but random forest and decision trees show the best result of less than 0.20% on adversarial examples. Anti-malware vendors retrain detectors after exploring such undetected examples, but MalGAN only needs one epoch retraining to obtain a 0% true positive rate. Kawai et al. later proposed some performance improvements [41].

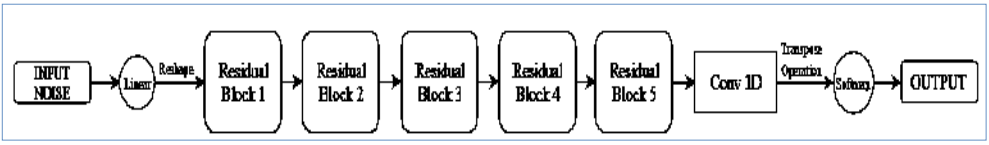


*Fig. 4. Generator architecture of PassGAN*

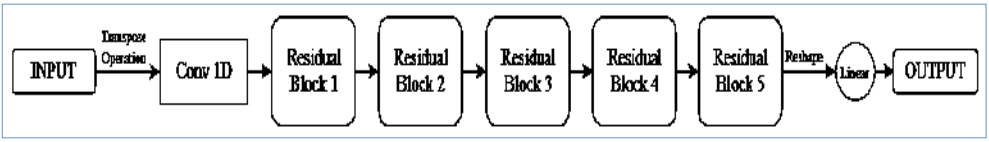


*Fig. 5. Discriminator architecture of PassGAN*

Another example use case for GAN in cybersecurity is the password guessing attack. There is a new way of generating password guesses based on DL and generative adversarial networks known as PassGAN [42]. The key difference in this approach is that NNs do not need a priori knowledge of the structure of passwords, in contrast to approaches based on rules, Markov models [43] and FLA [15]. PassGAN uses the improved training of Wasserstein GANs (IWGAN) of Gulrajani et al. [44] with the ADAM optimizer [45]. The generator and the discriminator in PassGAN are built from ResNets [46]. The architecture of the generator and the discriminator are shown in fig. 4 and fig. 5 [42], while residual block representation is shown in fig. 6 [42].

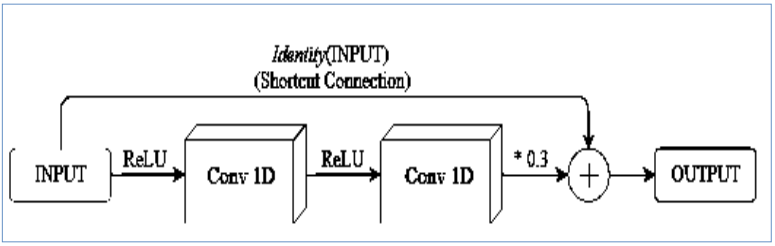


*Fig. 6. Architecture of residual block in PassGAN*

For maximum effectiveness, attackers most likely use several password-cracking tools, such as a HashCat [47], John the Ripper [48], PCFGs [49], OMEN [50] and FLA [15] to combine different attack methods. For example, by combining the output of PassGAN with the output of HashCat Best64 [51], researchers were able to guess between 51% and 73% additional unique passwords compared to HashCat [47] alone.

Traditional botnets wait for commands from the C&C, but now, attackers use automation to make decisions independently. Fortinet researchers predicted that cybercriminals will replace botnets with intelligent clusters of compromised devices called hivenets, a type of attack that is able to leverage peer-based self-learning to target vulnerable systems with minimal supervision [52].

In the initial stages of an attack, attackers often face the challenge of bypassing captcha. Suphannee et al. [53] designed a low-cost attack that uses DL technologies for the semantic annotations of images. The system requires about 19 seconds per challenge to solve challenges, with an accuracy of 70.78% for reCaptcha [54] and 83.5% for the Facebook image captcha. The system has to automatically identify which of the given images are semantically similar to the sample image. First, the system collects information for all the images through Google Reverse Images Search (GRIS) [55]; Clarifai [56], which is built on deconvolutional networks [57]; TDL [58], which is based on deep Boltzmann machines [59]; NeuralTalk [60] and Caffe [61]. Next,

if a hint is not provided, the system searches for the sample image in the labelled dataset to obtain one, if possible.

## *5. Fully ML-powered cyberattack*

As mentioned in the previous section, ML-powered cyberattacks are not a hypothetical future concept. This section describes how an automated cyberattack can be carried out using ML.

We considered two scenarios for the weaponization and delivery stages: First, in the case of humanless intrusion, attackers can use a similar tool but utilize information provided by Shodan [62] or Mitch [30] instead of features obtained using a computer vision. Second, attackers can use social engineering, using tools for profiling and for spear-phishing described in the previous section [34, 35] and creating click-bytes links to infect the victim [35, 36]. For automated exploit generation, adversaries can use open-sourced angr [63] framework developed by Shellphish and combine it with MalGAN to bypass defensive systems.

In the post-exploitation stage, attackers can guess stolen passwords using PassGAN [42]. The newest method is using intelligent evasion techniques proposed by Darktrace researchers [64] and further self-propagating with a series of autonomous decisions. It is also possible to turn infected systems into a hivenet [52].

As these examples demonstrate, ML can help hackers in every stage of the attack. With the advance level of development of the cybercriminal infrastructure, an advanced attack requires no hands-on-keyboard such as the case at present.

## *6. Conclusion*

When introducing an ML-based system, we should remember that ML is not a panacea. No system is safe. Under certain conditions, ML both protects vulnerabilities and creates new gaps. ML can be compared to a dog: 'Machine learning can do anything you could train a dog to do – but you're never totally sure what you trained the dog to do'.

We should also note the consequences that more active implementation of ML can bring: First, automation and the resulting loss of human jobs and second, inevitable conflict with the existing legal framework, for example, when using technologies to prevent cybercrime or cyberterrorism. In such a situation, the accused is implicated for crimes that have not yet been committed, which are not regulated by any legal norm. Moreover, some of the information learned by ML may be private or confidential, which violates laws in some countries. Similarly, poor quality or inadequate quantity of ML in the cybersecurity of data on predictions are based can lead to wrong decisions and irreparable mistakes.

## References

[1] P. Krensky, J. Hare. Hype Cycle for Data Science and Machine Learning, 2018. Gartner, 2018. Accessed: Sep. 10, 2019. [Online] Available at: https://www.gartner.com/en/documents/3883664/hype-cycle-for-data-science-and-machine-learning-2018.

[2] Nils J. Nilsson. Artificial Intelligence: A New Synthesis. Elsevier Inc, 1998, 513 p.

[3] Businesses recognize the need for AI & ML tools in cybersecurity. Helpnetsecurity.com. Accessed: Sep. 10, 2019. [Online] Available at: https://www.helpnetsecurity.com/2019/03/14/ai-ml-tools-cybersecurity/.

[4] T. Mitchell. Machine Learning. A Guide to Current Research. Tom M. Mitchell, Jaime G. Carbonell, Ryszard S. Michalski (Eds.). Springer Science & Business Media, 1986, 429 p.

[5] J. Grus. Data Science from Scratch: First Principles with Python. O'Reilly Media, 2015, 330 p.

[6] L. Deng, D. Yu. Deep Learning: Methods and Applications. Foundations and Trends in Signal Processing, vol. 7, nos. 3–4, 2014, pp. 199- 200

[7] K. Warr. Strengthening Deep Neural Networks: Making AI Less Susceptible to Adversarial Trickery. O'Reilly Media, Inc., 2019, 246 p.

[8] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen. Improved Techniques for Training GANs. arXiv:1606.03498, 2016.
[9] Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S.Ozair, A. Courville, Y. Bengio, Generative Adversarial Networks. arXiv:1406.2661, 2014.
[10] J. Han, J. Pei, M. Kamber. Data Mining: Concepts and Techniques. Morgan Kaufmann, 3rd edition, 2011, 744 p.
[11] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, R. Wirth. CRISPDM 1.0 step-by-step data mining guide. SPSS, 2000, 78 p.
[12] S. Dilek, H. Çakır, M. Aydın. Applications Of Artificial Intelligence Techniques To Combating Cyber Crimes: A Review. International Journal of Artificial Intelligence & Applications (IJAIA), vol. 6, vo. 1, 2015, pp. 21-39.
[13] S. Revathi and A. Malathi. A Detailed Analysis on NSL-KDD Dataset Using Various Machine Learning Techniques for Intrusion Detection. International Journal of Engineering Research and Technology, vol. 2, issue 12, 2013, pp. 1848-1853.
[14] L. Buczak and E. Guven. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. IEEE Communications Surveys & Tutorials, vol. 18, no. 2, 2016, pp. 1153–1176.
[15] W. Melicher, B. Ur, S.Segreti, S. Komanduri, L. Bauer, N. Christin, L. Cranor. Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks. In Proc. of the 25th USENIX Security Symposium, 2016, pp. 176-191.
[16] Ciaramella, P. D'Arco, A. De Santis, C. Galdi, R. Tagliaferri. Neural Network Techniques for Proactive Password Checking. IEEE Transactions on Dependable and Secure Computing, vol. 3, no. 4, 2006, pp. 327-339.
[17] Chris Brook. What is User and Entity Behavior Analytics? A Definition of UEBA, Benefits, How It Works, and More. Accessed: Oct. 10, 2019. [Online]. Available at: https://digitalguardian.com/blog/what-user-and-entity-behavior-analytics-definition-ueba-benefits-how-it-works-and-more
[18] Anna L. Buczak, Erhan Guven. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. IEEE Communications Surveys & Tutorials, vol. 18, no. 2, 2016, pp. 1153-1176.
[19] E. Kaspersky. Laziness, Cybersecurity, and Machine Learning. Accessed: Oct. 10, 2019. [Online]. Available: https://eugene.kaspersky.com/2016/09/26/laziness-cybersecurity-and-machine-learning/.
[20] J. Roberts. Cyber-Hunting at Scale (CHASE). Accessed: Oct. 19, 2019. [Online]. Available: https://www.darpa.mil/program/cyber-hunting-at-scale.
[21] Hernandez-Suarez, G. Sanchez-Perez, K. Toscano-Medina, V. Martinez-Hernandez, H. Perez-Meana, J. Olivares-Mercado, V. Sanchez. Social Sentiment Sensor in Twitter for Predicting Cyber-Attacks Using ℓ1 Regularization. Sensors, vol. 18, no. 5, 2018, pp. 1380.
[22] Caliskan, F. Yamaguchi, E. Dauber, R. Harang, K. Rieck, R. Greenstadt, A/ Narayanan. De-anonymizing Programmers via Code Stylometry. In Proc. of the 24th USENIX Security Symposium, 2015, pp. 255-270.
[23] Caliskan, F. Yamaguchi, E. Dauber, R. Harang, K. Rieck, R. Greenstadt, A. Narayanan. When Coding Style Survives Compilation: De-anonymizing Programmers from Executable Binaries. arXiv:1512.08546, 2015.
[24] S. Repalle, V. Kolluru. Intrusion Detection System using AI and Machine Learning Algorithm. International Research Journal of Engineering and Technology (IRJET), vol. 04, issue 12, 2017, pp. 1709-1715.
[25] J. Vacca, S. Ellis. Firewalls: Jumpstart for Network and Systems Administrators. Digital Press, 2004, 448 p.
[26] E. Ucar, E. Ozhan. The Analysis of Firewall Policy Through Machine Learning and Data Mining. Wireless Personal Communications, vol. 96, issue 2, 2017, pp. 2891 - 2909.
[27] S. Prandl, M. Lazarescu, D. Pham. A Study of Web Application Firewall Solutions. Lecture Notes in Computer Science, vol. 9478, 2015, pp. 501-510.
[28] Introduction to Forcepoint DLP Machine Learning. Accessed: Oct. 10, 2019. [Online]. Available at: https://www.websense.com/content/support/library/data/v84/machine_learning/machine_learning.pdf
[29] OWASP Top 10 - 2017 The Ten Most Critical Web Application Security Risks. Accessed: Nov. 5, 2019. [Online]. Available at: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

[30] S. Calzavara, M. Conti, R. Focardi, A. Rabitti, G. Tolomei. Mitch: A Machine Learning Approach to the Black-Box Detection of CSRF Vulnerabilities. In Proc. of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P), 2019, pp. 528-543.
[31] G. Pellegrino, M. Johns, S. Koch, M. Backes, C. Rossow. Deemon: Detecting CSRF with Dynamic Analysis and Property Graphs. arXiv:1708.08786, 2017.
[32] Z. Mao, N. Li, I. Molloy. Defeating Cross-Site Request Forgery Attacks with Browser-Enforced Authenticity Protection. Lecture Notes in Computer Science, vol. 5628, 2009, pp. 238-255.
[33] Philippe De Ryck, Lieven Desmet, Thomas Heyman, Frank Piessens. CsFire: Transparent Client-Side Mitigation of Malicious Cross-Domain Requests. In Proc. of the Second International Symposium on Engineering Secure Software and Systems, 2010, pp. 18-34.
[34] Jacob Wilkin. Mapping Social Media with Facial Recognition: A New Tool for Penetration Testers and Red Teamers. Accessed: Oct. 19, 2019. [Online]. Available at: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/mapping-social-media-with-facial-recognition-a-new-tool-for-penetration-testers-and-red-teamers/.
[35] R. Zellers, A. Holtzman, H. Rashkin, Y. Bisk, A. Farhadi, F. Roesner, Y. Choi. Defending Against Neural Fake News. arXiv:1905.12616, 2019.
[36] J. Seymour, P. Tully. Weaponizing data science for social engineering: Automated E2E spear phishing on Twitter. Accessed: Oct. 19, 2019. [Online]. Available at: https://www.blackhat.com/docs/us-16/materials/us-16-Seymour-Tully-Weaponizing-Data-Science-For-Social-Engineering-Automated-E2E-Spear-Phishing-On-Twitter-wp.pdf
[37] S. Thompson. Phight Phraud. Accessed: Nov. 6, 2019. [Online]. Available at: https://www.journalofaccountancy.com/issues/2006/feb/phightphraud.html
[38] M. Jakobsson, J. Ratkiewicz. Designing ethical phishing experiments: a study of (ROT13) rOnl query features. In Proc. of the 15th International Conference on World Wide Web, 2006, pp. 513-522.
[39] E. Bursztein, B. Benko, D. Margolis, T. Pietraszek, A. Archer, A. Aquino, A. Pitsillidis, S. Savage. Handcrafted fraud and extortion: Manual account hijacking in the wild. In Proc. of the 2014 Conference on Internet Measurement, 2014, pp. 347-358.
[40] W. Hu, Y. Tan. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. arXiv:1702.05983, 2017.
[41] M. Kawai, K. Ota, M. Dong. Improved MalGAN: Avoiding Malware Detector by Leaning Cleanware Features. In Proc. of the 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), 2019, pp. 40 - 45.
[42] Hitaj, P. Gasti, G. Ateniese, F. Perez-Cruz. PassGAN: A Deep Learning Approach for Password Guessing. arXiv:1709.00440, 2017.
[43] Narayanan, V. Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In Proc. of the 12th ACM Conference on Computer and Communications Security, 2005, pp. 364 - 372.
[44] Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A. Courville. Improved training of wasserstein GANs. In Proc. of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 5769-5779.
[45] Kingma, J. Ba. Adam: A Method for Stochastic Optimization. arXiv:1412.6980, 2017.
[46] K. He, X. Zhang, S. Ren, J. Sun. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2015.
[47] Hashcat – advanced password recovery. Accessed: Oct. 19, 2019. [Online]. Available at: https://hashcat.net/hashcat/
[48] John the Ripper password cracker. Accessed: Oct. 19, 2019. [Online]. Available at: https://www.openwall.com/john/
[49] M. Weir, S. Aggarwal, B. Medeiros, BGlodek. Password cracking using probabilistic context-free grammars. In Proc. of the 30th IEEE Symposium on Security and Privacy, 2009, pp. 391-405.
[50] M. Dürmuth, F. Angelstorf, C. Castelluccia, D. Perito, C. Abdelber. OMEN: Faster Password Guessing Using an Ordered Markov Enumerator. Lecture Notes in Computer Science, vol. 8978, 2015, pp. 119-132.
[51] hashcat/rules/best64.rule. Accessed: Nov. 10, 2019 [Online]. Available at: https://github.com/hashcat/hashcat/blob/master/rules/best64.rule.
[52] Derek Manky. Fortinet Predicts Highly Destructive and Self-learning “Swarm” Cyberattacks in 2018. Accessed: Nov. 10, 2019 [Online]. Available at: https://www.fortinet.com/corporate/about-us/newsroom/press-releases/2017/predicts-self-learning-swarm-cyberattacks-2018.html.

[53] S. Sivakorn, J. Polakis, A.Keromytis. I'm not a human: Breaking the Google reCAPTCHA. Accessed: Nov. 10, 2019 [Online]. Available at: https://www.blackhat.com/docs/asia-16/materials/asia-16-Sivakorn-Im-Not-a-Human-Breaking-the-Google-reCAPTCHA-wp.pdf.
[54] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. reCAPTCHA: Human-based character recognition via web security measures. Science, vol. 321, no. 5895, 2008, pp. 1465-1468.
[55] A. Krizhevsky, I. Sutskever, G. Hinton. ImageNet classification with deep convolutional neural networks. Communications of the ACM, June 2017, vol. 60, issue 6, pp. 84-90.
[56] Clarifia. Accessed: Nov. 10, 2019 [Online]. Available at: https://www.clarifai.com.
[57] M. Zeiler, G. Taylor, Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In Proc. of the International Conference on Computer Vision, 2011, pp. 2018-2025.
[58] Toronto Deep Learning Demos, Accessed: Nov. 10, 2019 [Online]. Available at: http://deeplearning.cs.toronto.edu.
[59] N. Srivastava, R. Salakhutdinov. Multimodal Learning with Deep Boltzmann Machines. Journal of Machine Learning Research, vol. 15, 2014, pp. 2949-2980
[60] Andrej Karpathy. Deep Visual-Semantic Alignments for Generating Image Descriptions. Accessed: Nov. 10, 2019 [Online]. Available at: https://cs.stanford.edu/people/karpathy/cvpr2015.pdf.
[61] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. arXiv:1408.5093, 2014.
[62] Shodan. Accessed: Oct. 19, 2019 [Online]. Available at: https://www.shodan.io/.
[63] Angr. Accessed: Oct. 19, 2019 [Online]. Available at: https://angr.io/.
[64] The Next Paradigm Shift AI-Driven Cyber-Attacks. Accessed: Oct. 19, 2019 [Online]. Available at: https://www.darktrace.com/en/resources/wp-ai-driven-cyber-attacks.pdf.

## Information about authors / Информация об авторах

Sergey Mikchailovitch AVDOSHIN – Candidate of Technical Science, Professor, Head of the School of Software Engineering at National Research University Higher School of Economics since 2005. Research interests include design and analysis of computer algorithms, simulation and modeling, parallel and distributed processing, deep Web, blockchain technology.

Сергей Михайлович АВДОШИН – кандидат технических наук, профессор, руководитель департамента программной инженерии факультета компьютерных наук НИУ ВШЭ с 2005 года. Сфера научных интересов: разработка и анализ компьютерных алгоритмов, имитация и моделирование, параллельные и распределенные процессы, теневой интернет, технология блокчейн.

Aleksandr LAZARENKO – head of R&D department of Group-IB, cybercrime investigator from 2015, author of scientific papers on the privacy, anonymity, security of blockchain projects.

Александр Вячеславович ЛАЗАРЕНКО – руководитель департамента инноваций и разработки продуктов Group-IB, расследует киберпреступления с 2015 года, автор научных работ по вопросам конфиденциальности, анонимности, безопасности блокчейн-проектов.

Nataliia Igorevna CHICHILEVA – junior specialist of R&D department Group-IB, student of System and Software Engineering master's programme at HSE. Her research interests include information security, discrete mathematics, optimizations problem and travelling salesman problem.

Наталия Игоревна ЧИЧИЛЕВА – младший специалист департамента инноваций и разработки продуктов Group-IB. В настоящее время также является студенткой магистерской программы «Системная и программная инженерия». Профессиональные интересы – информационная безопасность, дискретная математика, задача оптимизации, задача коммивояжера.

Pavel Andreevich NAUMOV – Junior Specialist in the Department of Innovation and Product Development Group-IB, a student with a degree in Computer Security at Moscow State

Technical University. His professional interests include mathematical methods of information protection, hardware and software reverse engineering, security of development and applications.

Павел Андреевич НАУМОВ – младший специалист департамента инноваций и разработки продуктов Group-IB, студент специалитета по программе «Компьютерная безопасность» в МГТУ им. Н.Э.Баумана. Профессиональными интересами являются математические методы защиты информации, программно-аппаратная обратная разработка, безопасность разработки и приложений.

Petr Georgievich KLYUCHAREV – Candidate of Technical Science, Associate Professor of the Department of Information Security, BMSTU. Research interests: cryptography, discrete mathematics, theoretical computer science, machine learning, software development.

Петр Георгиевич КЛЮЧАРЕВ – кандидат технических наук, доцент кафедры «Информационная безопасность» МГТУ им. Н. Э. Баумана. Сфера научных интересов: криптография, дискретная математика, теоретическая информатика, машинное обучение, разработка программного обеспечения.

**DOI: 10.15514/ISPRAS-2019-31(5)-16**



# Анализ корректности синхронизации компонентов ядра операционных систем

*П.С. Андрианов, ORCID: 0000-0002-6855-7919 <andrianov@ispras.ru>*

*Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*

**Аннотация.** Большинство современных инструментов статической верификации плохо масштабируются на сложное программное обеспечение. Целью работы была разработка инструмента, который станет золотой серединой между точными и медленными инструментами статической верификации и быстрыми, но менее точными инструментами статического анализа. Основной идеей подхода является абстракция от точного взаимодействия потоков и анализ каждого потока отдельно от всех остальных, но в некотором окружении, которое моделирует влияние потоков друг на друга. Окружение содержит описание возможных действий над разделяемыми данными и примитивами синхронизации, а также условий их применения. Варьируя точность построения окружения, можно добиваться необходимого баланса между скоростью и точностью анализа в целом. Формальное описание предлагаемого подхода было сделано с использованием теории адаптивного статического анализа. Это позволило сформулировать условия и доказать корректность предлагаемого подхода в этих условиях. Для эффективного поиска состояний гонки используется специальная модель памяти, которая позволяет разделять области памяти на непересекающиеся регионы, соответствующие типам данных. Реализация предложенного подхода во фреймворке CPAchecker позволяет переиспользовать существующие техники анализа с минимальными изменениями. А реализация дополнительных техник анализа в рамках предложенной теории позволяет повысить точность анализа. Результаты проведенных экспериментов на двух наборах тестовых задач позволяют заключить о масштабируемости и практической применимости метода.

**Ключевые слова:** состояние гонки; раздельный анализ потоков; статическая верификация; операционная система Linux

**Для цитирования:** Андрианов П.С. Анализ корректности синхронизации компонентов ядра операционных систем. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 203-232. DOI: 10.15514/ISPRAS-2019-31(5)-16

# Analysis of correct synchronization of operating system components

*P.S. Andrianov, ORCID: 0000-0002-6855-7919 <andrianov@ispras.ru>*

*Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*

**Abstract.** Most of the software model checker tools do not scale well on complicated software. Our goal was to develop a tool, which provides an adjustable balance between precise and slow software model checkers and fast and imprecise static analyzers. The key idea of the approach is an abstraction over the precise thread interaction and analysis for each thread in a separate way, but together with a specific environment, which models effects of other threads. The environment contains a description of potential actions over the shared data and synchronization primitives, and conditions for its application. Adjusting

the precision of the environment, one can achieve a required balance between speed and precision of the complete analysis. A formal description of the suggested approach was performed within a Configurable Program Analysis theory. It allows formulating assumptions and proving the soundness of the approach under the assumptions. For efficient data race detection we use a specific memory model, which allows to distinguish memory domains into the disjoint set of regions, which correspond to a data types. An implementation of the suggested approach into the CPAchecker framework allows reusing an existed approaches with minimal changes. Implementation of additional techniques according to the extended theory allows to increase the precision of the analysis. Results of the evaluation allow confirming scalability and practical usability of the approach.

**Keywords:** Data race; Thread-Modular approach; Software verification; Linux kernel

**For citation:** Andrianov P.S. Analysis of correct synchronization of operating system components. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019, pp. 203-232 (in Russian). DOI: 10.15514/ISPRAS-2019-31(5)-16

## *1. Введение*

Верификация многопоточных программ всегда являлась более сложной задачей, чем верификация последовательных программ. Точное вычисление всех возможных чередований (interleavings), приводит к комбинаторному взрыву числа состояний. Поэтому, большинство инструментов статической верификации применяют различные техники оптимизации: редукция частичных порядков (partial order reduction) [1,2], абстракция счетчика (counter abstraction) [3] и другие. Тем не менее, большинство современных инструментов статической верификации плохо масштабируются на промышленное программное обеспечение. Этот факт подтверждается результатами сравнения инструментов статической верификации на наборе задач SV-COMP [4]. Задачи из категории «многопоточность», основанные на драйверах операционной системы Linux, вызывают значительные сложности для всех инструментов статической верификации.

Одной из альтернатив методам проверки моделей являются методы статического анализа, которые нацелены на быстрый поиск ошибок без абсолютной уверенности в финальном вердикте. Такие инструменты применяют различные фильтры и эвристики для ускорения анализа и поэтому не могут гарантировать корректность, то есть отсутствие ошибок. В данной работе представлен подход к статической верификации многопоточного программного обеспечения, который позволяет выбирать баланс между скоростью и точностью проводимого анализа.

Суть предлагаемого подхода состоит в следующем. Поскольку объектом верификации будет большая многопоточная программа, мы заранее отказываемся от анализа всей программы с учетом всех возможных взаимодействий потоков и рассматриваем каждый поток по-отдельности. В этом случае состояния каждого потока становятся *частичными*, то есть они не содержат информацию о других потоках и, следовательно, не могут описать полное состояние всей программы. Возможное влияние потоков друг от друга аппроксимируется сверху множеством действий, которое потоки могут совершать над разделяемыми данными, в том числе примитивами синхронизации.

Таким образом, аппроксимация возможных действий, или *эффектов*, формируется одновременно для всех потоков и называется *окружением*. Несмотря на то, что окружение является единым для всех потоков, отсюда не следует, что все входящие в него эффекты будут применены к некоторому потоку, так как для каждого из эффектов определяются условия его применения, которые зависят от используемого анализа. Кроме того, в условия применения эффекта могут включаться требования на конкретные операторы и состояния потока.

Точность построения окружения определяет во многом точность и скорость работы всего инструмента. Точность анализа можно повысить, комбинируя различные техники анализа. Для реализации этой идеи была использована платформа CPAchecker [5,6],

которая предоставляет богатый набор техник верификации. Анализ с раздельным рассмотрением потоков (thread-modular approach) [7-10] также можно реализовать как одну из техник, которая встраивается в CPAchecker и пополняет традиционный набор техник верификации таких как, например, CEGAR [11] и предикатная абстракция [12].

Эффективное расширение платформы CPAchecker требует не просто добавления еще одного вида анализа. В идеале нужно, чтобы максимальное число видов анализа могли работать одновременно и обмениваться данными между собой. Необходимым условием такой тесной интеграции является либо следование уже определенной теории CPA, либо модификация имеющейся теории таким образом, что старая теория оказывалась частным случаем новой. Именно такая задача ставилась в данном исследовании.

Поиск состояний гонки обычно состоит из двух основных этапов:

1. построение множества достижимых состояний;
2. нахождение в построенном множестве специальных состояний, образующих состояние гонки.

Эти два шага могут выполняться последовательно или параллельно в зависимости от инструмента статической верификации. Например, некоторые инструменты статической верификации при добавлении каждого следующего состояния проверяют, не образует ли оно состояние гонки с некоторым уже достижимым ранее состоянием анализа. И в случае обнаружения ошибки, такой анализ останавливает свое выполнение. Однако, такой способ является слишком медленным и непрактичным для поиска состояний гонки, хотя он успешно применяется, например, для решения задач достижимости или поиска ошибок, связанных с некорректным использованием памяти в последовательных программах.

Инструменты статического анализа, которые ищут потенциальные состояния гонки, обычно используют Lockset алгоритм для поиска таких ошибок. В предложенном подходе используется более точный алгоритм, в котором потенциальное состояние гонки является парой *совместных* переходов, которые модифицируют одну и ту же память. Совместность здесь означает, что два частичных состояния двух потоков могут быть частью одного глобального состояния. Таким образом, если рассматривать только абстракцию над примитивами синхронизации, это сводится к алгоритму Lockset, но при использовании других вариантов анализа, является более точным. Предикатная абстракция вместе с моделью памяти BnB [14-16] позволяет значительно улучшить работу с доступами по указателю и позволяет сохранить корректность при разумных предположениях.

Оценка подхода производилась на множестве задач, основанных на драйверах операционной системы Linux. Они были подготовлены с помощью системы Klever, которая позволяет проводить верификацию больших программных систем [17, 18]. Klever разделяет большой объем кода на отдельные фрагменты – верификационные задачи – и подготавливает для них модель окружения.

Основным вкладом данной работы является:

1. развитие теории CPA, которая позволяет комбинировать технику thread-modular с другими подходами, такими как предикатная абстракция;
2. реализация предложенной теории в инструменте CPAchecker, который был успешно апробирован на множестве задач, основанных на драйверах операционной системы Linux.

Статья организована следующим образом. В разд. 2 представлены основные сложности современных инструментов статической верификации и основы предлагаемого подхода. В разд. 3 представлена основная идея подхода. Разд. 4 вводит основные определения и модель программы. Следующие 7 разделов посвящены описанию расширения теории CPA: разд. 6 описывает thread-modular подход в терминах CPA, разд. 7 – 12 содержат расширенное описание основных анализов (CPA). В разд. 13 описаны основные

особенности поиска состояний гонки в предлагаемом подходе. В разд. 14 представлены результаты работы инструмента на наборе SV-COMP и драйверах операционной системы Linux. В разд. 15 представлен краткий обзор родственных работ.

## *2. Пример запуска существующих инструментов верификации*

Рассмотрим пример верификационной задачи[1] из SV-COMP'19 [4]. Эта верификационная задача основана на реальном состоянии гонки[2]. Файл с исходным кодом содержит более 7 000 строк кода и 4 создаваемых потока: один поток для базовых функций platform устройства, один – для обработки прерываний, один – для функций power management и один начальный поток, который выполняет операции инициализации-деинициализации модуля. Все примитивы синхронизации ядра (мьютексы и спинлоки) были заменены на pthread мьютексы. Известная ошибка была записана, как задача достижимости, следующим образом:

```
tmp = tspi->rst;
assert(tmp == tspi->rst);
```

Подробные результаты работы инструментов могут быть найдены на официальном сайте SV-COMP[3]. В основном, все современные инструменты верификации столкнулись с проблемами:

- CBMC: «pointer handling for concurrency is unsound – UNKNOWN»;
- CPAchecker: «Unsupported feature: BDD-analysis does not support arrays»;
- SMACK: «Exception thrown in lockpwn»;
- yogar-cbmc: «out of memory»;
- Ultimate: «Ultimate could not prove your program».

Основным вызовом для инструментов верификации в этом примере стало большое количество операций в потоках, большая часть из которых выполнялась над разделяемыми данными. Это означало, что потоки могли сильно влиять друг на друга. Это приводит к следующим подпроблемам, которые обычно игнорируются при анализе небольших «учебных» программ:

1. Анализ многопоточных программ должен быть достаточно точным, чтобы выдавать как можно меньшее количество ложных предупреждений, но быть достаточно эффективным, чтобы решать реальные задачи. Многие эффективные виды анализа не поддерживают сложные структуры данных (например, BDD анализ [19], анализ явных значений [20]). И наоборот, точные подходы вызывают проблемы при анализе длинных путей с переключениями между потоками (например, анализ предикатов [21], подход с ограничиваемой проверкой модели [22]).
2. Эффективное представление примитивов синхронизации. Многие подходы кодируют блокировки как переменные, которые атомарно проверяются и присваиваются (например, [8, 10]). Кодированные таким образом блокировки смешиваются с другими переменными и усложняют общий анализ.

Кроме того, задача поиска состояния гонки в верификационных задачах записывается как задача достижимости, что является подсказкой для инструмента верификации, какой доступ к памяти может содержать ошибку. На практике верификатор не знает точное местоположение потенциального состояния гонки, и поэтому он должен проверять все

[1] https://github.com/sosy-lab/sv-benchmarks.git, sv-benchmarks/c/ldv-linux-3.14-races/linux-3.14–drivers–spi–spi-tegra20-slink.ko.cil.i

[2] https://patchwork.kernel.org/patch/9915305

[3] https://sv-comp.sosy-lab.org/2019/results/results-verified/META ConcurrencySafety.table.html

возможные доступы к памяти. Это еще более усложняет задачу. В данной работе представлен разработанный метод, который позволяет решать такие задачи.

## *3. Схема предлагаемого метода*

Как уже было сказано, в методах поиска состояний гонок условно можно выделить два этапа, точнее, две фазы анализа: построение множества достижимых состояний и проверка требований, в данном случае, поиск парных состояний, образующих гонку. Анализ программы может проводиться путем последовательного чередования этих фаз или путем их параллельного выполнения. Заметим, что проверка требований обычно требует не очень больших затрат, так как представляет собой условие на полученные состояния. Примерами таких проверок могут быть: отсутствие состояний специального вида (задача достижимости), отсутствие пары состояний специального вида (задача поиска гонок). Построение множества достижимых состояний, напротив, порождает ряд проблем, в первую очередь, связанных с эффективностью.

Рассмотрим простую программу, в которой всего два потока (рис. 1).

```
volatile int g = 0;
volatile int d = 0;
Thread1 {
1:  g = 1;
2:  d = 1;
3:  …
}
Thread2 {
4:  if (d == 1) {
5:    g = 2;
6:  }
}
```

*Рис. 1. Пример небольшой программы*
*Fig. 1. An example of a small program*

Это некоторый модельный пример, в котором используется конструкция неявной синхронизации между потоками: первый поток инициализирует некоторые данные (в данном случае, глобальную переменную g), а затем выставляет флаг, что данные готовы. Второй поток может использовать эти данные только после выставления флага, поэтому в этом примере нет состояния гонки для переменной g. Классические методы проверки моделей перебирают все возможные варианты чередования двух потоков (пример одного из возможных вариантов приведен на рис. 2).

| Thread 1 | Thread 2 |
|---|---|
| g = 1; | |
| d = 1; | |
| | [d == 1] |
| | g = 2; |

*Рис. 2. Пример одного варианта выполнения потоков*
*Fig. 2. An example of an execution*

С точки зрения инструмента статической верификации, необходимо рассмотреть полное множество состояний программы, которые возникают при всех возможных чередованиях (рис. 3).

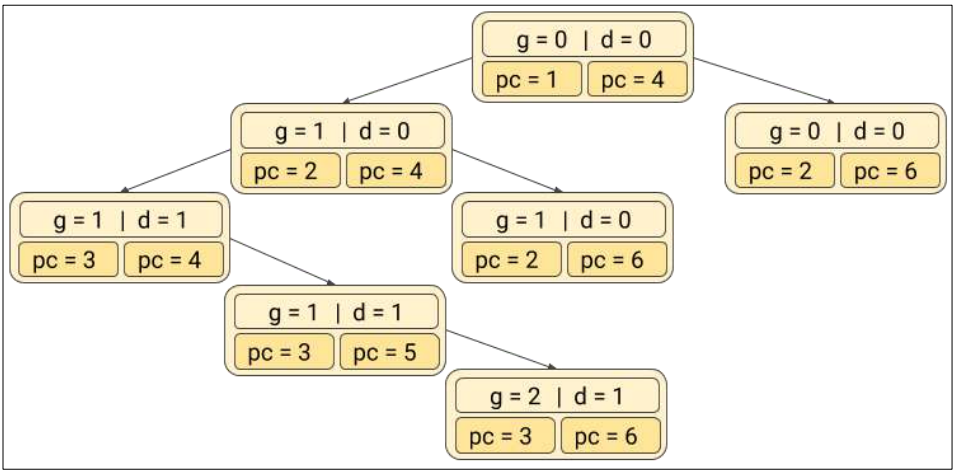


*Рис. 3. Построение множества чередований*
*Fig. 3. Construction of interleaving set*

Даже в простом примере и при различных оптимизациях общее число состояний растет с катастрофической скоростью. Происходит так называемый «комбинаторный взрыв» числа состояний, что приводит к исчерпанию ресурсов. Таким образом, классические методы проверки моделей не могут обеспечить доказательства корректности программы.

Простые методы статического анализа пытаются вычислить аппроксимацию сверху возможных действий одного потока на другой, так называемый эффект потока. Однако, они не способны прослеживать сложные зависимости между переменными. Например, зависимости между глобальными переменными, которые, в свою очередь, могут быть модифицированы из других потоков. В общем случае, это требует вычисления некоторой неподвижной точки, что является нежелательным при статическом анализе, так как значительно возрастают требования к ресурсам. В итоге, в таких сложных случаях считается, что глобальные переменные могут принимать любые значения. А это, в свою очередь, снижает точность анализа.

Предлагаемый подход базируется на известной идее раздельного анализа потоков (thread-modular approach). Потоки в этом случае анализируются по-отдельности, одновременно с этим строится общее для всех потоков окружение, которое аппроксимирует сверху влияние других потоков. Это окружение формируется на основе анализа всех потоков, так как каждый поток являются частью окружения для других потоков. Для каждого потока определяется, как и в каких условиях он может модифицировать разделяемые данные, использовать примитивы синхронизации и выполнять иные действия, влияющие на другие потоки. Точность анализа потока зависит от того, как точно будет сформировано окружение. Однако, остается вопрос как эффективно вычислять и представлять окружение.

При анализе последовательных программ успешной техникой, позволяющей уменьшить число рассматриваемых состояний программы, является абстракция. Она позволяет абстрагироваться от несущественных деталей программы и рассматривать обобщенные (абстрактные) состояния, которые могут соответствовать целому множеству реальных (конкретных) состояний программы. Это позволяет значительно сократить пространство состояний.

Ключевой идеей предлагаемого подхода является расширение абстракции не только на состояния программы, но и на операции, то есть, переходы потока. Настраивая уровень абстракции, можно получать варианты анализа, которые будут ближе к статическому анализу многопоточных программ, или к классической статической верификации.

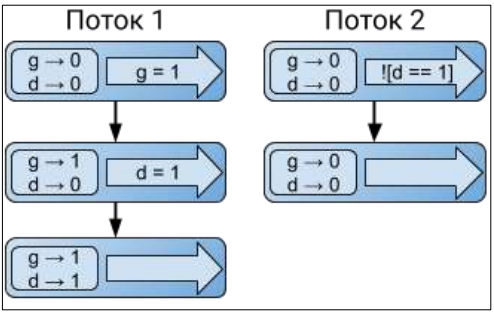


*Рис. 4. Построение абстрактных переходов двух потоков*
*Fig. 4. Construction of abstract states of the first thread*

Рис. 4 показывает часть абстрактного графа достижимости (Abstract Reachability Graph, ARG) для первого и второго потока без влияния друг на друга. Представленный анализ основан на простом анализе явных значений [20], который отслеживает только явные значения переменных. Переход содержит в себе абстрактное состояние и абстрактную операцию. Первое абстрактное состояние содержит информацию только о значении глобальной переменной x. Новая информация о значении переменной y появляется в дочерних элементах, после того как выполнен переход, соответствующей инициализации переменной.

Теперь необходимо учесть влияние потоков друг на друга, то есть сформировать окружение. Будем называть *проекцией* операции потока описание ее эффекта, видимого для других потоков. Например, любые модификации локальных переменных потока не влияют на другие потоки, то есть их проекция является пустой операцией. Модификация глобальной переменной является значимой для всех потоков, поэтому ее проекция должна совпадать с самой операцией, либо аппроксимировать ее сверху, например, теряя информацию о точном присваиваемом значении. При этом в проекции может быть не только информация о самом действии, но и об условии на его применение к другому потоку. Например, на рисунке 5 первый поток, присваивая “g = 1” меняет значение переменной g с нуля на единицу. Можно представить проекцию этой операции таким образом: если значение переменной x равно нулю, то оно может быть изменено на единицу. Иными словами, проекция состоит из двух частей: условия ее применения ([g == 0]) и непосредственно действия (g → 1).

При анализе некоторого потока одновременно строится его представление для остальных в качестве окружения. Оно состоит из набора проекций операций этого потока. Далее каждая из этих проекций должна быть применена ко всем возможным (с учетом условий внутри проекций) состояниям других потоков. Что, в свою очередь, может породить новые, еще не исследованные состояния, а значит, и проекции.

После построения переходов в потоках независимо друг от друга (рис. 4), для всех переходов вычисляются проекции. Для второго потока, например, это действие, которое меняет значение переменной x значение с нуля на тройку. Остальные действия второго потока не модифицируют глобальные переменные и не порождают значимых проекций. Затем эта проекция применяется к каждому состоянию первого потока. На рисунке 5 приведен результат применения к первому переходу. Также эту проекцию можно применить и ко второму, однако, никаких новых путей это не породит. К третьему переходу первого потока применить данную проекцию нельзя, так как значение переменной x не удовлетворяет условию проекции. Применение проекции к первому переходу порождает новый путь выполнения, который в свою очередь может породить новые проекции.

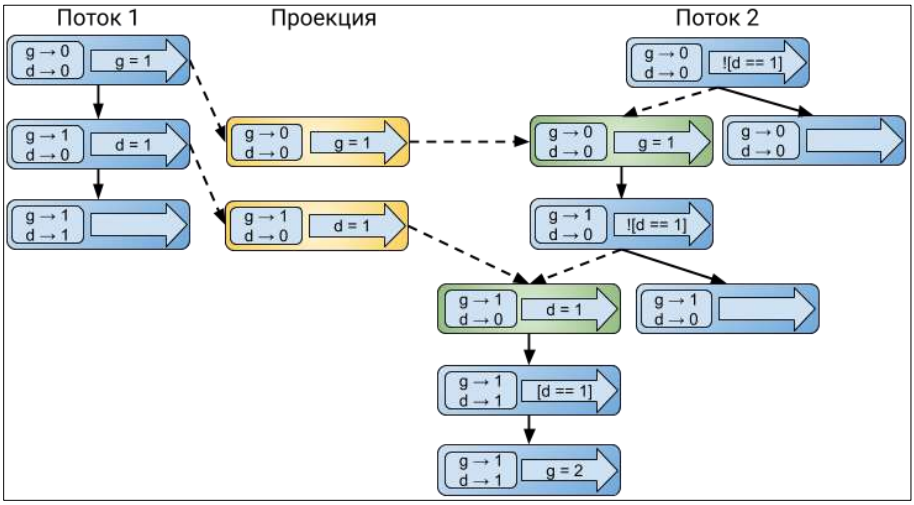


*Рис. 5. Построение абстрактных переходов для двух потоков*
*Fig. 5. Construction of abstract transitions for both of the threads*

На рис. 5 представлен вариант с точными проекциями, которые рассматривают переходы другого потока так, как они есть. На рисунке представлены не все возможные проекции и порожденные ими переходы. Например, отсутствует проекция перехода «g = 2» второго потока.

Для проверки возможности состояния гонки нам необходимо найти два перехода, которые модифицируют одну переменную: «g = 1» в первом потоке и «g = 2» во втором. Далее, необходимо проверить, являются ли два абстрактных состояния совместными, то есть, могут ли они быть частью одного глобального состояния. В данном случае, в частичных состояниях потока значения глобальных переменных имеют разные значения, а значит, они не могут быть частью одного глобального состояния, то есть, указанные два перехода не могут быть выполнены одновременно. Отсюда следует, что состояние гонки отсутствует.

Предлагаемый подход предоставляет гибкие варианты конфигурации для решения каждой конкретной задачи. Как было показано на примерах, проекции действий потока могут быть представлены более точной абстракцией или, наоборот, слишком общей. Проекции нескольких операций могут быть объединены в одну или быть рассмотрены по-отдельности. Это позволяет выбирать необходимый баланс между точностью и скоростью. Рассмотрим другой вариант построения абстрактных переходов на рис. 6.

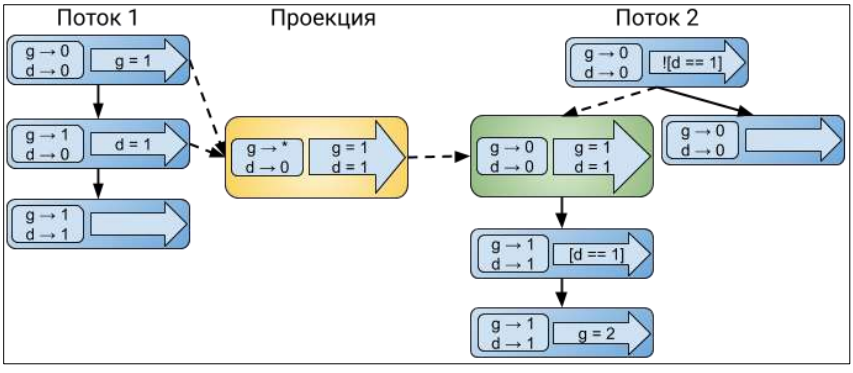


*Рис. 6. Построение абстрактных переходов для двух потоков*
*Fig. 6. Construction of abstract transitions for both of the threads*

Здесь используется более абстрактное представление проекции, при котором несколько воздействий потока объединяются в одну проекцию (эффект от окружения). При этом обычно теряется некоторая информация. В частности, в данном случае была потеряна информация о точном значении переменной g, и поэтому данный объединенный эффект может применяться при любых ее значениях. Это позволяет сократить число состояний для анализа.

Пример показывает, как анализ рассматривает эффекты влияния одного потока на другой, что гарантирует корректность подхода. Более того, он показывает гибкость подхода, который позволяет варьировать уровень абстракции, например, на стадии построения проекции, выбирая уровень абстракции каждого перехода в окружении.

## *4. Основные определения*

В этом разделе представлены основные определения параллельной программы и достижимых конкретных состояний программы, необходимые для описания математической модели параллельной (многопоточной) программы и теории CPA.

Математической моделью параллельной программы будет программа на простом императивном языке, в котором имеются только такие операторы, которые оказывают эффект на другие потоки или сами зависят от эффектов, оказываемых другими потоками. Такими операторами служат операторы присваивания, проверки условия (ветвления), примитивы синхронизации, создания потоков. Формальный аппарат описания таких моделей – это теория, для построения которой воспользуемся простым императивным языком, достаточно выразительным, чтобы описывать модели, упомянутые выше. Обозначим множество операций в модели программе Ops.

Параллельная программа представляется автоматом потока управления (Control Flow Automaton, CFA [5]), который состоит из множества L точек программы (моделируются программным счетчиком, pc) и множеством $G \subseteq L \times Ops \times L$ дуг (ребер) потока управления (моделируют операции, которые выполняются, когда управление переходит от одной точки в программе к другой). Операция создания потока создает новый поток с идентификатором из множества T и этот поток начинает свое выполнение из некоторой точки программы из L. Множество переменных программы, которые встречаются в операторах присваивания и условиях из Ops обозначаются X, а их значения ограничим множеством целых чисел $\mathbb{Z}$. Подмножества X, содержащие только локальные и глобальные переменные, обозначаются $X^{local}$ и $X^{global}$ соответственно. Операции захвата/освобождения примитивов синхронизации определяются на множестве переменных-блокировок S, которые имеют значения из $T \cup \{\bot_T\}$, где $t \in T$ означает, что соответствующая блокировка была захвачена потоком t, а $\bot_T$ означает, что соответствующая блокировка не была захвачена.

*Конкретным состояниям* программы называется четверка ($c_{pc}$ , $c_l$ , $c_g$ , $c_s$ ), где

1. отображение $c_{pc}$: $T \to L$ является частичной функцией из идентификаторов потока в точку программы, в которой находится этот поток;
2. отображение $c_l$: $T \to C^{local}$ является частичной функцией из идентификаторов потока в присваивание локальным переменным их значений, то есть $C^{local} : X^{local} \to \mathbb{Z}$;
3. отображение $c_g$: $X^{global} \to \mathbb{Z}$ является присваиванием значений глобальным переменным;
4. отображение $c_s$: $S \to T \cup \{\bot_T\}$ является присваиванием значений переменным блокировки.

Множество конкретных состояний программы обозначается как C. Отображения $c_{pc}$ и $c_l$ представляют собой локальную часть состояния потока, а $c_g$ и $c_s$ – глобальную. Обозначим dom – домен частичной функции, например, $dom(c_l) = \{t | \exists (t, \cdot) \in c_l\}$.

Для каждого состояния $c=(c_{pc},c_l,c_g,c_s) \in C$ должно выполняться условие $dom(c_{pc})=dom(c_l)$, означающее, что домены локальных частей состояния должны быть консистентны и содержать одинаковое число потоков. Это множество обозначим $dom(c)=dom(c_{pc})=dom(c_l)$.

Определим отношение переходов $\xrightarrow{g,t} \subseteq C \times G \times T \times C$, где дуга $g \in G$, а поток $t \in T$.

Определим множество конкретных переходов $\top = C \times G \times T$. Элемент $\tau \in \top$ – это тройка $\tau = (c, g, t)$. Будем писать $\tau_1 \rightarrow \tau_2$ если $\exists c_3 \in C$: $c_1 \xrightarrow{g_1,t_1} c_2 \xrightarrow{g_2,t_2} c_3$. Семантику операций определим позже. В любом случае корректный переход $g = (l,\cdot,l') \in G$ должен удовлетворять следующим условиям:

1. переход начинается в состоянии c: $t \in dom(c) \wedge c_{pc}(t) = l$.
2. программный счетчик потока t переходит к точке $l'$: $t \in dom(c') \wedge c'_{pc}(t) = l'$.
3. каждый переход в потоке t может изменить локальные части только этого же потока, а локальные состояния других потоков должны остаться без изменений; заметим, что для программ с указателями это не верно, но пока мы не рассматриваем такие программы в теории.

Будем обозначать $Reach_{\rightarrow}(\tau) = \{\tau | \exists \tau_1, \ldots, \tau_n \in \top, .\tau \rightarrow \tau_1 \rightarrow \ldots \rightarrow \tau_n = \tau_0\}$. Будем обозначать *eval(c,t,expr)* значение выражения *expr* над переменными из $X^{local} \cup X^{global}$ со значениями из состояния $c \in C$ потока $t \in T$.

## 4.1 Операторы условия

Для дуги проверки условия $g = (l, assume(expr), l') \in G, t \in T, l, l' \in L$ переход $c \xrightarrow{g,t} c', c = (c_{pc}, c_l, c_g, c_s), c' = (c'_{pc}, c'_l, c'_g, c'_s) \in C$ существует, если

- $dom(c) = dom(c')$, то есть переход не меняет множества потоков;
- $c_l = c'_l, c_g = c'_g, c_s = c'_s$, то есть переход не меняет значений переменных;
- $c_{pc}(t) = l, c'_{pc}(t) = l'$, то есть переход соответствует общим условиям на начало и конец;
- $eval(c, t, expr) \neq 0$, то есть значения переменных потока удовлетворяют проверяемому условию.

## 4.2 Операторы присваивания

Для дуги присваивания $g = (l, assign(y, expr), l') \in G, t \in T, l, l' \in L$ переход $c \xrightarrow{g,t} c', c = (c_{pc}, c_l, c_g, c_s), c' = (c'_{pc}, c'_l, c'_g, c'_s) \in C$ существует, если:

- $dom(c) = dom(c')$, то есть переход не меняет множества потоков;
- $\forall x \in X^{local}, t' \in T. c'_l(t')(x) = \begin{cases} c_l(t')(x), \text{если } x \neq y \vee t \neq t' \\ eval(c, t, expr), \text{если } x = y \wedge t = t' \end{cases}$;
- $\forall x \in X^{global}. c'_g(x) = \begin{cases} c_g(x), \text{если } x \neq y \\ eval(c, t, expr), \text{если } x = y \end{cases}$;
- $c_s = c'_s$, множество блокировок не меняется при обычном присваивании.
- $c_{pc}(t) = l, c'_{pc}(t) = l'$, то есть переход соответствует общим условиям на начало и конец.

## 4.3 Операции над примитивами синхронизации

Определим операции над примитивами синхронизации *acquire/release*. Предполагаем, что операция *acquire(s)* в потоке $t \in T$, где $s \in S$ – это специальная переменная блокировки, имеет стандартную семантику: атомарная проверка значения переменной s и, в случае

если s=$\perp_T$, присвоение ей идентификатора текущего потока. Например, такая же семантика рассматривается в [8].

Для дуги захвата блокировки $g = (l, acquire(s), l') \in G, t \in T, l, l' \in L$ переход $c \xrightarrow{g,t} c', c = (c_{pc}, c_l, c_g, c_s), c' = (c'_{pc}, c'_l, c'_g, c'_s) \in C$ существует, если:

- $dom(c) = dom(c')$, то есть переход не меняет множества потоков;
- $c_l = c'_l, c_g = c'_g$, то есть переход не меняет значений переменных;
- $c_{pc}(t) = l, c'_{pc}(t) = l'$, то есть переход соответствует общим условиям на начало и конец;
- $c_s(s) = \perp_T \wedge c'_s(s) = t$ , $\forall s' \in S: s' \neq s \Rightarrow c'_s(s') = c_s(s')$.

Операция освобождения блокировки *release(s)* является обратной к операции захвата блокировки и присваивает значение s = $\perp_T$, если эта блокировка была захвачена текущим потоком, то есть s = t.

Для дуги освобождения блокировки $g = (l, release(s), l') \in G, t \in T, l, l' \in L$ переход $c \xrightarrow{g,t} c', c = (c_{pc}, c_l, c_g, c_s), c' = (c'_{pc}, c'_l, c'_g, c'_s) \in C$ существует, если:

- $dom(c) = dom(c')$, то есть переход не меняет множества потоков;
- $c_l = c'_l, c_g = c'_g$, то есть переход не меняет значений переменных;
- $c_{pc}(t) = l, c'_{pc}(t) = l'$, то есть переход соответствует общим условиям на начало и конец;
- $c_s(s) = t \wedge c'_s(s) = \perp_T, \forall s' \in S: s' \neq s \Rightarrow c'_s(s') = c_s(s')$.

## 4.4 Операция создания потока

Определим семантику операции *thread_create($l_v$)* таким образом, что текущий поток переходит в следующую точку программы, а новый поток создается с идентификатором v ∈ T и начинает свое выполнение из $l_v$ ∈ L.

В общем случае возможно неограниченное создание потоков в программе, если, например, операция создания потока встречается в цикле.

Для дуги создания потока $g = (l, thread\_create(l_v), l') \in G, t \in T, l, l' \in L$ переход $c \xrightarrow{g,t} c', c = (c_{pc}, c_l, c_g, c_s), c' = (c'_{pc}, c'_l, c'_g, c'_s) \in C$ существует, если:

- $v \notin dom(c) \wedge dom(c') = dom(c) \cup \{v\}$, поток $v$ добавляется во множество потоков;
- $c_l = c'_l, c_g = c'_g, c_s = c'_s$, то есть переход не меняет значений переменных текущего потока;
- $c_{pc}(t) = l, c'_{pc}(t) = l'$, то есть переход соответствует общим условиям на начало и конец;
- $c'_{pc}(v) = l_v$, то есть новый поток начинает свое выполнение из своего начального состояния

Пока мы не рассматриваем операции ожидания потока(*thread_join (*v)), так как они будут усложнять объяснение и доказательство основной теоремы корректности предложенного подхода. Тем не менее, их поддержка также может быть добавлена.

## *5. Адаптивный статический анализ с абстрактными переходами*

В классической теории адаптивного статического анализа (Configurable Program Analysis, CPA) [5, 6], абстрактное состояние представляет множество конкретных состояний программы. В расширенной теории, абстрактное состояние является частичным и может не представлять никакое состояние программы. Вот почему функция конкретизации, которая сопоставляет абстрактные состояния с конкретными, в расширенной теории

отличается от классической. В частности, она определяется на множестве абстрактных состояний.

Как следствие, абстрактный переход также является частичным. Поэтому анализ не может гарантировать, что последующие конкретные переходы будут достижимы за один шаг оператора transfer. В общем случае для этого может понадобиться k шагов. Для подхода с раздельным анализом потоков k = 2: анализ выполняет обычный переход в потоке, а затем распространяет его на все остальные потоки в качестве перехода в окружении. Это требует двух итераций алгоритма.

Определим формально адаптивный статический анализ с абстрактными переходами $\mathbb{D} = (D, \Pi, merge, stop, prec, \rightsquigarrow)$. Он состоит из абстрактного домена *D*, множество точности *П*, оператор слияния *merge*, оператор останова *stop*, оператор настройки точности *prec*, и отношение переходов $\rightsquigarrow$.

1. Абстрактный домен $D = (\top, \mathcal{E}, [[\cdot]])$ определяется множеством $\top$ конкретных переходов, $\top \subseteq C \times G \times T$, полурешеткой $\mathcal{E}$ над абстрактными переходами и функцией конкретизации [[.]]. Полурешетка $\mathcal{E} = (E, \bot, \top, \sqsubseteq, \sqcup)$ состоит из (возможно бесконечного) множества E абстрактных элементов, верхнего элемента решетки $\top \in E$, нижнего элемента решетки $\bot \in E$, частичный порядок $\sqsubseteq \subseteq E \times E$ и функцию объединения $\sqcup$: $E \times E \to E$. Функция объединения определяет минимальный элемент решетки, который больше заданных элементов.

Функция конкретизации $[[\cdot]] : 2^E \to 2^\top$ для каждого множества абстрактных переходов $R \subseteq E$ определяет множество соответствующих конкретных переходов программы. Основным отличием от классической функции конкретизации – это определение на множестве абстрактных элементов. Таким образом, $\forall R \subseteq E : [[R]] \supseteq \bigcup_{e \in R} [[\{e\}]]$. Это означает, что суммарное знание от множества абстрактных переходов может быть больше, чем объединение знания от каждого частичного перехода.

2. Множество точности П определяет возможную точность абстрактного домена. Анализ использует элементы точности, чтобы отслеживать различные абстрактные состояния с различной точностью. Пара $(e, \pi)$ называется абстрактным элементом e с точностью $\pi$. Операторы на абстрактном домене параметризованы точностью.

Для $R \subseteq E \times \Pi$ будем обозначать $[[R]] = [[\bigcup_{(e,\pi) \in R} \{e\}]]$.

3. Отношение переходов $\rightsquigarrow$: $E \times \Pi \times 2^E \times E$ определяет для каждого частичного перехода $\hat{e}$ с точностью $\pi$ возможный следующий переход e'. При этом результат может зависеть от множества достижимых элементов $R \subseteq E$. Будем обозначать $(\hat{e}, \pi) \rightsquigarrow^R e$ if $(\hat{e}, \pi, R, e) \in \rightsquigarrow$.

Определим множество $Reach^k$ по индукции: $\forall R \subseteq E : Reach^0(R) = R$

$$\forall k \geq 1 : Reach^{k+1}(R) = \bigcup_{e \in Reach^k(R)} \{e' \mid e \rightsquigarrow e'\} \cup Reach^k(R)$$

Основное требование к оператору *transfer* является аппроксимация сверху множества конкретных переходов:

$$\exists k \geq 1 : \forall R \subseteq E : Reach^k(R) = \bigcup_{\tau \in [[R]]} \{\tau' \mid \tau \to \tau'\}$$

Это требование ослабляет требование на *transfer* классической теории. Оно означает, что оператор должен выдавать соответствующие конкретные переходы на после одного шага, как в классической версии, а после k шагов. Для подхода с раздельным анализом потоков k = 2, как мы увидим в дальнейшем.

4. Оператор слияния *merge*: $E \times E \times \Pi \to E$ ослабляет второй параметр, используя информацию от первого параметра и возвращает новый абстрактный элемент с точностью, соответствующей третьему параметру. Оператор *merge* должен удовлетворять следующему условию:

$$\forall e, e' \in E, \pi \in \Pi : e' \sqsubseteq merge(e, e', \pi)$$

5. Оператор останова *stop*: $E \times 2^E \times \Pi \times \Pi \rightarrow \{true, false\}$ проверяет, покрывается ли абстрактный элемент, данный в качестве первого параметра, множеством элементов, данных как второй параметр. Оператор останова может, например, искать среди множества элементов такой, который покрывает ($\sqsubseteq$) данный элемент. Оператор останова должен удовлетворять следующим условиям:

$$\forall e, e' \in E, \pi \in \Pi : stop(e, R, \pi) \Rightarrow \forall \hat{R} \subseteq E : [[\{e\} \cup \hat{R}]] \subseteq [[R \cup \hat{R}]]$$

6. Функция настройки точности *prec*: $E \times \Pi \times 2^E \times \Pi \rightarrow E \times \Pi$ вычисляет новый абстрактный элемент и новую точность для заданного элемента с точностью и множества абстрактных элементов. Функция настройки точности может выполнять ослабление (расширение) абстрактного элемента вместе с изменением точности. Функция настройки точности должна удовлетворять следующему требованию:

$$\forall e, e' \in E, \pi, \pi' \in \Pi, R \subseteq E \times \Pi : (e', \pi') = prec(e, \pi, R) \Rightarrow e \sqsubseteq e'$$

В целом, множество точности П, оператор останова stop, оператор объединения merge, оператор настройки точности prec остаются такими же, как и в классической теории CPA. На рис. 7 представлен основной алгоритм, который вычисляет множество достижимых абстрактных переходов, также не изменяется за исключением расширения оператора *transfer*.

```
waitlist := {(e0, π0)};
reached := {(e0, π0)};
while waitlist ≠ ∅ do
    pop (e, π) from waitlist;
    for e' in (e, π) ⇝(reached) e') do
        (ê, π̂) = prec(e', π, reached);
        for each (e'', π'') ∈ reached do
            e_new = merge(ê, e'', π̂);
            if e_new ≠ e'' then
                waitlist := waitlist \ {(e'', π'')} ∪ {(e_new, π'')};
                reached := reached \ {(e'', π'')} ∪ {(e_new, π'')};
            end
        end
        if !stop(ê, reached, π̂) then
            waitlist := waitlist ∪ {(ê, π̂)};
            reached := reached ∪ {(ê, π̂)};
        end
    end
end
```

*Рис. 7. Основной алгоритм CPA(D, $e_0$ , $\pi_0$)*
*Fig. 7. The main algorithm CPA(D, $e_0$ , $\pi_0$)*

Для этого алгоритма основная теорема может быть доказана даже с ослабленными требованиями. Доказательство повторяет доказательство классической теоремы.

**Теорема** (корректность). Для заданного адаптивного статического анализа с абстрактными переходами $\mathbb{D}$, начального состояния $e_0$ с точностью $\pi_0$ алгоритм CPA(D, $e_0$ , $\pi_0$) вычисляет множество абстрактных переходов, которое аппроксимирует сверху множество достижимых конкретных переходов:

$$[[CPA(\mathbb{D}, e_0, \pi_0)]] \supseteq Reach_{\rightarrow}([[\{e_0\}]])$$

Следует отметить, что на практике используется одновременно несколько различных CPA для анализа исходного кода. При этом структура множества CPA напоминает древовидную, где корень этого дерева предоставляет операторы для основного алгоритма, представленного на рис. 7. Различные CPA взаимодействовать друг с другом для повышения точности анализа. Так, в разд. 6 будет представлен верхнеуровневый CPA,

который требует наличие вложенного CPA. Примеры вложенных CPA будут представлены в разделах 7 – 11. CPA, представленный в разд. 12, реализует параллельную композицию нескольких вложенных анализов. Разд. 13 описывает, как реализуется поиск состояний гонки при помощи этих инструментов.

Некоторые служебные CPA (CallstackCPA, AutomatonCPA), не реализующие никакие техники анализа, могут быть применены без изменений по сравнению с классической версией теории, и поэтому не будут описываться далее. CPA, которые реализуют различные техники анализа (анализ явных значений, анализ предикатов и др.) необходимо доработать для того, чтобы они могли поддерживать переходы потоков в окружении, в том числе реализовать оператор проекции. Эти CPA будут описаны в соответствующих разделах.

## *6. ThreadModularCPA*

В этом разделе представлен CPA, который реализует логику подхода с раздельным анализом потоков. Основная функциональность ThreadModularCPA заключается в вычислении для каждого перехода в потоке потенциального эффекта для окружения, то есть проекцию этого перехода, а также в применении полученных эффектов окружения на соответствующий поток.

ThreadModularCPA включает в себя некоторый внутренний анализ с расширенным множеством операторов, которые необходимы для работы с окружением: оператор проекции, оператор совместности, и оператор композиции. Сначала формально определим расширение внутреннего CPA.

### 6.1 Расширение внутреннего CPA

Определение CPA для подхода с раздельным анализом потоков расширяется тремя новыми операторами: $\mathbb{I} = (D_I, \Pi_I, I, \text{merge}_I, \text{stop}_I, \text{prec}_I, \text{compatible}_I, \cdot|_p, \text{compose}_I)$.

Абстрактный домен $D_I = (\top_I, \mathcal{E}_I, \oplus_I)$ состоит из множества конкретных переходов $\top_I$, полурешетки $\mathcal{E}_I$, и оператора композиции частичных состояний $\oplus_I$. Таким образом, внутренний анализ должен определить не функцию конкретизации $[[\cdot]]$, а оператор композиции $\oplus$, так как подход с раздельным анализом потоков требует одинаковой схемы вычисления конкретных состояний.

Как было уже сказано, состояния и переходы являются частичными, поэтому они могут не соответствовать напрямую конкретным состояниям и переходам. Чтобы получить полный переход, нужно взять композицию множества частичных переходов, которые соответствуют всем доступным потокам. Совместные частичные переходы могут быть объединены в полный конкретный переход с помощью оператора композиции $\oplus: E \times T \times 2^{E \times T} \to 2^{\top}$. Он возвращает множество конкретных переходов, которое соответствует данным частичным переходам.

$\oplus$ должен соответствовать полурешетке. Так, если один из абстрактных переходов меньше, чем другой, то композиция с тем же множеством не должна получить большее множество конкретных переходов.

Оператор проверки совместности $\text{compatible}_I$: $E \times E \to \{\text{true}, \text{false}\}$ проверяет, могут ли два частичных перехода начинаться из общего полного родительского состояния.

Оператор проекции $\cdot|_p$: $E \to E$ проецирует переход в потоке на другой поток. Например, проекция может содержать модификации глобальных переменных, но опускать изменения локальных данных для потока.

$\text{compose}_I$: $E \times E \to E$ объединяет два абстрактных перехода в один. Он применяет абстрактную дугу из одного перехода к абстрактному состоянию другого перехода.

В дальнейшем мы будем использовать оператор $\text{apply}_\text{I}$, как комбинацию трех операторов: $\cdot|_\text{p}$, $\text{compose}_\text{I}$ и $\text{compatible}_\text{I}$:

$$\forall e, e' \in E: \boldsymbol{apply}(e, e') = \begin{cases} \boldsymbol{compose_I}(e, e'|_p), \text{если } \boldsymbol{compatible_I}(e, e'|_p) \\ \bot, \text{иначе} \end{cases}$$

Таким образом, оператор $\text{apply}_\text{I}$ означает, что переходы могут быть объединены, только если они совместны. Результатом применения оператора является новый переход, который будем называть переходом в окружении, так как он представляет собой эффект окружения.

## 6.2 CPA для раздельного анализа потоков

Определим специальный CPA, который реализует логику раздельного анализа потоков: $\mathbb{TM} = (\text{D}_\text{TM}, \Pi_\text{TM}, \rightsquigarrow_\text{TM}, \text{merge}_\text{TM}, \text{stop}_\text{TM}, \text{prec}_\text{TM})$, который основан на внутреннем CPA $\mathbb{I} = (\text{D}_\text{I}, \Pi_\text{I}, \text{I}, \text{merge}_\text{I}, \text{stop}_\text{I}, \text{prec}_\text{I}, \text{compatible}_\text{I}, \cdot|_\text{p}, \text{compose}_\text{I})$.

Абстрактный домен $\text{D}_\text{TM} = (\top, \mathcal{E}, [[\cdot]])$, множество конкретных переходов $\top = \top_\text{I}$, а решетка $\mathcal{E} = \mathcal{E}_\text{I}$. Функция конкретизации $[[\cdot]]$ выражается через оператор композиции $\bigoplus_\text{I}$:

$$\forall R \subseteq E: [[R]] = \bigcup_{k=1}^{\infty} \bigcup_{\substack{e_0, e_1, \ldots, e_k \in R \\ t_0, t_1, \ldots, t_k \in T}} \bigoplus_I \left(\begin{matrix} e_0 \\ t_0 \end{matrix}, \left\{\begin{matrix} e_1 \\ t_1 \end{matrix}, \ldots, \begin{matrix} e_k \\ t_k \end{matrix}\right\}\right)$$

Отношение переходов определяет следующие переходы, после чего применяются все достигнутые переходы, как переходы в окружении, к новым переходам, а новые переходы, как переходы в окружении, – к уже достижимым (рис. 8).

```
result := ∅ ;
for each ê : e0 ⇝^R_I ê do
    result := result ∪ {ê} ;
    for each e' ∈ reached do
        result := result ∪ {apply(e', ê)} ;
        result := result ∪ {apply(ê, e')} ;
    end
end
return result
```

*Рис. 8. $\text{transfer}_\text{TM}(e_0, \pi_0, \text{reached})$*
*Fig. 8. $\text{transfer}_\text{TM}(e_0, \pi_0, \text{reached})$*

Множество точности Π, операторы *merge*, *stop*, *prec* соответствуют операторам внутреннего CPA.

## *7. LocationCPA*

В этом разделе представлен простой анализ точек программы (Location Analysis) $\mathbb{L} = (\text{D}_\text{L}, \Pi_\text{L}, \rightsquigarrow_\text{L}, \text{merge}_\text{L}, \text{stop}_\text{L}, \text{prec}_\text{L}, \text{compatible}_\text{L}, \cdot|_\text{p}, \text{compose}_\text{L})$, который отслеживает абстрактные точки программы. Анализ расширяет классический LocationCPA для возможности применения его вместе с ThreadModularCPA.

1. Абстрактный домен $\text{D}_\text{L} = (\top_\text{L}, \mathcal{E}_\text{L}, \bigoplus_\text{L})$. Абстрактный переход состоит из абстрактного состояния $s \in E_\text{L}^S$, и абстрактной дуги $q \in E_\text{L}^\text{T}$. $E_\text{L}^S$ – это множество абстрактных точек программы (англ. program location), которые отображаются на конкретные точки программы в CFA с помощью функции $loc: E_\text{L}^S \to 2^L$. $\top_L^S$ означает, что анализ не знает, в какой именно точке программы находится анализ. Более формально, $loc(\top_L^S) = L$. В общем случае анализ может использовать абстрактными точками программы, которые соответствуют нескольким конкретным точкам программы, но в дальнейшем будет

описан простой вариант анализа, который рассматривает только одиночные точки программы:

$$\forall \mathrm{s} \in E_{\mathrm{L}}^{S}: \mathrm{s} = \top_{L}^{S} \vee \mathrm{s} = \bot_{\mathrm{L}}^{\mathrm{S}} \vee \ loc(\mathrm{s}) = \{\mathrm{l}\} \in \mathrm{L}$$

В этом случае используемая решетка $\mathcal{E}_{\mathrm{L}}^{S}$ является плоской, то есть любые два невырожденных состояния (неравные $\top_{L}^{S}$ или $\bot_{\mathrm{L}}^{\mathrm{S}}$) несравнимы.

Абстрактная дуга основана на обычной CFA дуге и содержит только начальную точку программы и конечную: $E_{\mathrm{L}}^{T} \subseteq E_{\mathrm{L}}^{S} \times E_{\mathrm{L}}^{S}$.

2. Множество точности является вырожденным и содержит только один элемент: $\Pi_S = \{\emptyset\}$.

3. Переход $e \rightsquigarrow_{\mathrm{L}} e', e = (s, g), e' = (s', g'), q = (pred, suc), q' = (pred', suc')$ существует, если изменение точки программы соответствует абстрактной дуге. Более формально, $e \rightsquigarrow_P e' \Leftrightarrow loc(s) \cap loc(pred) \neq \emptyset, \exists g' \in G: g' = (l'_1, op, l'_2) \wedge l'_1 \in loc(pred') \cap loc(s') \wedge l'_2 \in loc(suc') \wedge s' = suc$.

4. Оператор слияния *merge* не объединяет элементы: $merge_{\mathrm{L}}(e, e', \pi) = e'$.

5. Оператор останова *stop* рассматривает уникальные состояния: $stop_{\mathrm{L}}(e, R, \pi) = (e \in R)$.

6. Точность не регулируется: $prec_{\mathrm{L}}(e, \pi, R) \ = (e, \pi)$.

7. Переход в одном потоке никак не влияет на переход в другом: $\forall e \in E_{\mathrm{L}}, e = (s, g): e|_P = (s, \varepsilon)$.

8. $\forall e, e' \in E_{\mathrm{L}}, e = (s, q): compose_{\mathrm{L}}(e, e') = \tilde{\mathrm{e}} = (s, \hat{q})$, где $\hat{q} = (s, s)$, так как переход в одном потоке никак не может повлиять на положение другого потока.

9. $\forall e_1, e_2 \in E_{\mathrm{L}}: compatible_{\mathrm{L}}(e_1, e_2) \equiv true$, так как два потока могут быть совместны в любых точках программы.

## *8. ThreadCPA*

Определим простой анализ потоков $\mathbb{T}$ = ($\mathrm{D_T}$, $\Pi_{\mathrm{T}}$, $\rightsquigarrow_{\mathrm{T}}$, $\mathrm{merge_T}$, $\mathrm{stop_T}$, $\mathrm{prec_T}$, $\mathrm{compatible_T}$, $\cdot|_{\mathrm{p}}$, $\mathrm{compose_T}$), который отслеживает идентификаторы потока.

Анализ потоков наследует ограничения из [7] и ограничен программами с конечным числом созданий потоков. Пусть в программе есть конечное число потоков, которые идентифицируются точками программы, в которых они начинают свое выполнение, то есть $T \subseteq L$ и для каждого оператора создания потока *thread_create(*$pc_v$*)* всегда создается поток с идентификатором $pc_v$. Заметим, что остальные типы анализов не ограничены числом созданий потоков, более того, возможно применение более сложного анализа потоков, который будет поддерживать неограниченное число созданий. Таким образом, общая теория поддерживает неограниченное число созданий потоков.

1. Абстрактный домен $\mathrm{D_T}$ = ($\top_{\mathrm{T}}$, $\mathcal{E}_{\mathrm{T}}$, $\oplus_{\mathrm{T}}$) основан на плоской решетке над множеством потоков T, где $\mathcal{E}_{\mathrm{T}} = \mathcal{E}_{\mathrm{T}}^{S} \times \mathcal{E}_{\mathrm{T}}^{\mathrm{T}}$. Множество абстрактных состояний $E_T^S = T \cup \{\bot_T^{\mathrm{S}}, \top_T^S\}$, в котором любые два невырожденных состояния (неравные $\top_T^S$ или $\bot_T^{\mathrm{S}}$) несравнимы. Множество абстрактных дуг содержит множество обычных CFA дуг и пустой переход в окружении: $E_{\mathrm{T}}^{\mathrm{T}} = \{\bot_{\mathrm{T}}^{\mathrm{T}}, \varepsilon, \top_{\mathrm{T}}^{\mathrm{T}}\} \cup G$.

2. Множество точности является вырожденным и содержит только один элемент: $\Pi_{\mathrm{T}} = \{\emptyset\}$.

3. Переход $e \rightsquigarrow_{\mathrm{T}} e', e = (s, g), e' = (s', g'), \mathrm{g} = (\cdot, \mathrm{op}, \cdot)$ существует, если

- $\mathrm{op} \neq$ *thread_create(*$pc_v$*)*, $s' = s, g' \in \mathrm{G}$, то есть при любой операции, кроме создания потока, состояние не меняется;
- $\mathrm{op} =$ *thread_create(*$pc_v$*)*, $s' = pc_v \vee s' = s, g' \in \mathrm{G}$. При создании потока создаются два дочерних состояния: одно соответствует созданному потоку, а другое – родительскому.

4. Оператор слияния *merge* не объединяет элементы: $merge_{\mathrm{T}}(e, e', \pi) = e'$.

5. Оператор останова *stop* рассматривает уникальные состояния: $stop_{\mathrm{T}}(e, R, \pi) = (e \in R)$.

6. Точность не регулируется: $prec_{\mathrm{T}}(e, \pi, R) = (e, \pi)$.

7. Переход в одном потоке никак не влияет на переход в другом: $\forall e \in E_{\mathrm{T}}, e = (s, g)\colon e|_P = (s, \varepsilon)$. Переходы в окружении ($\varepsilon$-переходы) не могут изменить идентификатор другого потока. Тем не менее, проекция влияет на совместность состояний, чтобы переходы в потоке не применялись, как эффекты окружения, к этому же самому потоку.

8. $\forall e, e' \in E_{\mathrm{T}}, e = (s, q), e' = (s', q')\colon compose_{\mathrm{T}}(e, e') = \tilde{e} = (s, q')$.

9. $\forall e_1, e_2 \in E_{\mathrm{T}}, e_1 = (s_1, q_1), e_2 = (s_2, q_2)\colon compatible_{\mathrm{T}}(e_1, e_2) = s_1 \neq s_2$, так как два состояния могут быть совместны, если они относятся к разным потокам.

## *9. ValueCPA*

Определим анализ явных значений $\mathbb{V}$ = ($\mathrm{D_V}$, $\Pi_{\mathrm{V}}$, $\rightsquigarrow_{\mathrm{V}}$, $\mathrm{merge_V}$, $\mathrm{stop_V}$, $\mathrm{prec_V}$, $\mathrm{compatible_V}$, $\cdot|_{\mathrm{P}}$, $\mathrm{compose_V}$), который отслеживает явные значения переменных. Он состоит из следующих элементов.

1. Абстрактный домен $\mathrm{D_P}$ = ($\top_{\mathrm{V}}$, $\mathcal{E}_{\mathrm{V}}$, $\oplus_{\mathrm{V}}$). $\mathcal{E}_{\mathrm{V}}$ = ($\mathrm{E_V}$, $\bot_{\mathrm{V}}$, $\top_{\mathrm{V}}$, $\sqsubseteq_{\mathrm{V}}$, $\sqcup_{\mathrm{V}}$). Абстрактный переход состоит из абстрактного состояния $\mathrm{s} \in E_V^S$, а абстрактная дуга $\mathrm{q} \in E_V^{\mathrm{T}}$, таким образом, $\mathrm{E_V} = E_{\mathrm{V}}^S \times E_{\mathrm{V}}^{\mathrm{T}}$, а $\mathcal{E}_{\mathrm{V}} = \mathcal{E}_{\mathrm{V}}^S \times \mathcal{E}_{\mathrm{V}}^{\mathrm{T}}$.

Абстрактное состояние этого анализа является отображением из имен переменных в их значение: $\forall \mathrm{s} \in E_V^S, \mathrm{s}\colon X \mapsto Z$, где $Z = \mathbb{Z} \cup \{\bot_Z, \top_Z\}$. Таким образом, множество абстрактных состояний является плоской решеткой над целыми числами. Верхний элемент решетки $\top_{\mathrm{V}}^S = \{v \mid \forall x \in X\colon v(x) = \top_Z\}$. является отображением, в котором любая переменная имеет любое значение. А нижний элемент решетки $\bot_{\mathrm{V}}^S = \{v \mid \exists x \in X\colon v(x) = \bot_Z\}$ является отображением, в котором никакая переменная не может иметь никакого явного значение. Такое состояние является недостижимым при реальном выполнении программы. Порядок является тривиальным: любые два невырожденных состояния (неравные $\top_{\mathrm{V}}^S$ или $\bot_{\mathrm{V}}^{\mathrm{S}}$) несравнимы.

Множество абстрактных дуг содержит множество обычных CFA дуг и переходы в окружении, которые определяются изменением глобальных переменных: $E_{\mathrm{V}}^{\mathrm{T}} = 2^{X \mapsto Z} \cup G$.

2. Точность анализа явных значений определяется отслеживаемыми переменными, таким образом множество точности содержит подмножества из всех переменных программы: $\Pi_{\mathrm{V}} = 2^X$.

3. Отношение перехода $e \rightsquigarrow_{\mathrm{V}} e', e = (s, g), e' = (s', g')$.

I. $g \in G, g = (\cdot, op, \cdot)$.

a. $g = (\cdot, assume(expr), \cdot)$:

$$\forall x \in X\colon s'(x) == \begin{cases} \bot_Z, \text{если } \nexists c.\, (x \to c)\colon (\mathrm{expr} \neq 0)_{/s} \\ c, \text{если } \exists!\, c.\, (x \to c)\colon (\mathrm{expr} \neq 0)_{/s} \text{ или } s(x) = c \\ \top_Z, \text{иначе} \end{cases}$$

Здесь $\mathrm{expr}_{/v}$ означает интерпретацию выражения expr над переменными из $X$ для абстрактного присваивания $v$. А выражение $(x \to c)\colon (\mathrm{expr} \neq 0)_{/s}$ означает, что значение $c$ у переменной $x$ удовлетворяет интерпретации.

b. $g = (\cdot, assign(w, expr), \cdot)$:

$$\forall x \in X\colon s'(x) = \begin{cases} \mathrm{expr}_{/s}, \text{если x = w} \\ s(x), \text{иначе} \end{cases}$$

c. В остальных случаях состояние не меняется $s = s'$.

II. $g: X \mapsto \mathcal{Z}$, это означает, что мы имеем переход, который меняет определенные переменные. В этом случае, следующее состояние

$$\forall x \in X: s'(x) = \begin{cases} g(x), \text{если x} \in \mathrm{dom}(g) \\ s(x), \text{иначе} \end{cases}$$

4. Оператор слияния *merge* не объединяет элементы: $merge_{\mathrm{V}}(e, e', \pi) = e'$.

5. Оператор останова *stop* рассматривает уникальные состояния: $stop_{\mathrm{V}}(e, R, \pi) = (e \in R)$.

6. Функция настройки точности вычисляет новое абстрактное состояние и точность, ограничивая присваивания только теми переменными, которые содержатся в точности: $prec_{\mathrm{V}}(e, \pi, R) = (e_{|\pi}, \pi)$.

7. Переход в окружении может затрагивать только глобальные переменные: $\forall e \in E_{\mathrm{V}}, e = (s, g): e|_P = (s^{global}, g^{global})$. Здесь отображение $s^{global}$ означает только ту часть, которая относится к глобальным переменным.

8. $\forall e, e' \in E_{\mathrm{V}}, e = (s, q), e' = (s', q'): compose_{\mathrm{V}}(e, e') = \tilde{e} = (s, q')$.

9. $\forall e_1, e_2 \in E_{\mathrm{V}}, e_1 = (s_1, q_1), e_2 = (s_2, q_2): compatible_{\mathrm{V}}(e_1, e_2) = \forall x \in X^{global}: (x \in dom(s_1) \wedge x \in dom(s_2)) \Rightarrow (s_1(x) \sqsubseteq s_2(x) \vee s_2(x) \sqsubseteq s_1(x))$, что означает консистентность значений глобальных переменных.

## *10. PredicateCPA*

В этом разделе описан известный анализ предикатов (Predicate Analysis) [21] с абстрактными переходами. Переходы анализа предикатов состоят из двух частей: абстрактного состояния и абстрактной дуги, которая может быть выражена либо обычной CFA дугой, либо логической формулой, кодирующей выполняемую операцию. Кроме того, локальные переменные в этих формулах должны быть переименованы для избежания коллизии имен для разных потоков.

Пусть $^{P}$ – это множество формул над переменными программы в теории без кванторов. Пусть P $\subseteq {}^{P}$ – это множество предикатов, а $v$: X $\rightarrow \mathbb{Z}$ – это отображение из переменных в их значения. Определим $v \vDash \varphi$, где $v$ называется моделью формулы $\varphi$.

Определим переименование переменных θ:X $\rightarrow$ X’ из пространства имен X в X’, которое применимо к формулам θ($\varphi$). Обозначим

$$\theta_{X,i} = \begin{cases} \text{x} \longmapsto \text{x} & \text{i, если x} \in \text{X} \\ \text{x} \longmapsto \text{x, иначе} \end{cases}, \text{ и } \theta_{\text{X,i}}^{-1} = \begin{cases} \text{x} & \text{i} \longmapsto \text{x, если x} \in \text{X} \\ \text{x} \longmapsto \text{x, иначе} \end{cases}$$

Обозначим $(\varphi)^{\pi}$ декартову предикатную абстракцию формулы φ.

Обозначим $SP_{op}(\varphi)$ – сильнейшее постусловие формулы φ и операции op.

Определим анализ предикатов $\mathbb{P}$ = ($D_P$, $\Pi_P$, $\rightsquigarrow_P$, $merge_P$, $stop_P$, $prec_P$, $compatible_P$, $\cdot|_P$, $compose_P$), который отслеживает выполнимость предикатов над переменными программы. Он состоит из следующих компонентов:

1. Абстрактный домен $D_P$ = ($\top_P$, $\mathcal{E}_P$, $\oplus_P$). $\mathcal{E}_P$ = ($E_P$, $\bot_P$, $\top_P$, $\sqsubseteq_P$, $\sqcup_P$). Абстрактный переход состоит из абстрактного состояния s $\in E_P^S$, а абстрактная дуга q $\in E_P^{\mathrm{T}}$, таким образом, $E_P$ = $E_P^S \times E_P^{\mathrm{T}}$, а $\mathcal{E}_P$ = $\mathcal{E}_P^S \times \mathcal{E}_P^{\mathrm{T}}$.

$E_P^S$ = $^{P}$, таким образом, состояние является формулой первого порядка. Верхний элемент решетки является тождественно истинной формулой $\top_P$ = true, а нижний элемент – тождественно ложной: $\bot_P$ = false. Частичный порядок $\sqsubseteq_P^S \subseteq E_P^S \times E_P^S$ определяется как e $\sqsubseteq_P^S$ $e' \Leftrightarrow$ e $\Rightarrow$ e′. Оператор слияния $\sqcup_P^S: E_P^S \times E_P^S \rightarrow E_P^S$ определяет ближайший элемент в соответствии с частичным порядком.

Абстрактная дуга q $\in E_P^{\mathrm{T}}$ – это действие, которое может быть выражено или формулой, или обычной CFA дугой: $E_P^{\mathrm{T}} = E_P^S \cup G$.

Не будем приводить формальное определение $\oplus_P$, так как оно требует достаточно много места. Основная идея состоит в том, что он возвращает множество конкретных переходов, которые соответствуют формуле (сильнейшему постусловию). Наиболее сложная часть определения это проверка, может ли множество частичных переходов соответствовать единственному глобальному переходу. Для перехода в потоке $e_0$ с обычной CFA дугой $q_0 \in G$ и переходов в окружении $e_1, \ldots, e_n$ проверка состоит из двух частей:

a) для абстрактного состояния $s_0$ из $e_0$ и всех состояний из переходов в окружении $s_1, \ldots, s_n$ существует общая модель v;
b) дуга $q_p$ из проекции $e_0|_p$ покрывается абстрактными дугами $q_j$ переходов в окружении $q_p \sqsubseteq q_j$.

Формально,

$$\forall \boldsymbol{e_0, e_1, \ldots, e_n} \in \boldsymbol{E_P}, \boldsymbol{e_i} = (\boldsymbol{s_i}, \boldsymbol{q_i}), \boldsymbol{s_i} \in \boldsymbol{E_P^S}, \boldsymbol{q_i} \in \boldsymbol{E_P^T}$$
$$\boldsymbol{C_{check}}(\boldsymbol{e_0}, \{\boldsymbol{e_1}, \ldots, \boldsymbol{e_n}\}) = \exists \boldsymbol{v} : \boldsymbol{v} \vDash \boldsymbol{s_0} \wedge \boldsymbol{\theta_{X^{local},1}}(\boldsymbol{s_1}) \wedge \cdots \wedge \boldsymbol{\theta_{X^{local},n}}(\boldsymbol{s_n}) \wedge$$
$$(\boldsymbol{q_0} \in \boldsymbol{G} \wedge \forall \boldsymbol{0} < \boldsymbol{j} \leq \boldsymbol{n}: \boldsymbol{q_j} \notin \boldsymbol{G} \wedge \boldsymbol{e_0}|_{\boldsymbol{p}} = (\boldsymbol{s_p}, \boldsymbol{q_p}): \boldsymbol{q_p} \sqsubseteq \boldsymbol{q_j}$$

2. Множество точности $\Pi_P = 2^P$ определяет точность абстрактного состояния как множество предикатов.

3. Оператор объединения *merge* может иметь несколько модификаций, например,

$\text{merge}_{\text{Join}}$ объединяет обе части перехода:

$$\forall e, e' \in E_P, \pi \in \Pi_p, e = (s, g): \quad merge_p(e, e', \pi) = \begin{cases} (s \vee s', g), \text{если } g = g', g \in G \\ e', \text{если } g \in G \wedge g' \in \mathcal{P} \vee g \in \mathcal{P} \wedge g' \in G \\ (s \vee s', g \vee g'), \text{если } g, g' \in \mathcal{P} \end{cases}$$

$\text{merge}_{\text{Eq}}$ объединяет только абстрактные дуги при равных (или покрытых) состояниях.

$\text{merge}_{\text{Sep}}$ не объединяет элементы.

4. Оператор останова *stop* проверяет, покрывается ли переход e некоторым другим переходом в множестве достижимости: $stop_P(e, R, \pi) = \exists e' \in R : (e \sqsubseteq e')$.

5. Функция настройки точности *prec* вычисляет предикатную абстракцию над предикатами в точности $\pi$: $prec_P(e, \pi, R) = e^\pi = (s^\pi, q)$.

6. Отношение переходов $e \rightsquigarrow_P e', e = (s, g), e' = (s, \cdot)$. Так как анализ предикатов не отслеживает релевантные дуги, он возвращает все возможные.

Для g ∈ G существует переход $e \rightsquigarrow_P e'$ с g = (·, op, ·), если $s' = (SP_{op}(e))^\pi$.

Для $g = \varphi \in \mathcal{P}$ следующее абстрактное состояние имеет вид $s' = \hat{\varphi} \wedge e$.

7. Определение проекции:

$$\forall e \in E_P, e = (s, g): e|_P = \begin{cases} e, \text{если } g \notin G \\ \left(\theta_{X^{local},env}(s), \theta_{X^{local},env}\left(SP_{op}(s)\right)\right), \text{иначе} \end{cases}$$

Проекция определяет, как переход выглядит в качестве окружения потока. Локальные переменные переименовываются для избежания коллизии имен. Поэтому только предикаты над глобальными переменными остаются значимыми. Состояние (первая часть перехода) представляет абстракцию над глобальной частью состояния потока. А дуга (вторая часть перехода) соответствует конкретной операции над глобальными переменными.

8. $\forall e, e' \in E_P, e = (s, q), e' = (s', q'): compose_P(e, e') = \tilde{e} = (s, q')$

9. $\forall e_1, e_2 \in E_P, e_i = (s_i, q_i), s_i \in E_P^S: compatible_P(e_1, e_2) = \exists v: v \vDash \theta_{X^{local},1}(s_1) \wedge \theta_{X^{local},2}(s_2)$

Проверка совместности означает, что переходы могут быть объединены в один глобальный. А это невозможно, если глобальные предикаты неконсистентны, то есть не существует общей модели для двух частичных формул.

## *11. LockCPA*

Определим анализ примитивов синхронизации $\mathbb{S}$ = ($D_S$, $\Pi_S$, $\rightsquigarrow_S$, $merge_S$, $stop_S$, $prec_S$, $compatible_S$, $\cdot|_S$, $compose_S$), который отслеживает множество захваченных блокировок (переменных синхронизации) для каждого потока. Он состоит из следующих компонентов:

1. Абстрактный домен $D_S$ = ($\top_S$, $\mathcal{E}_S$, $\oplus_S$). $\mathcal{E}_S = \mathcal{E}_S^S \times \mathcal{E}_S^T$. $\mathcal{E}_S^S = 2^S \cup \{\perp_S, \top_S\}$ – это множество всех подмножеств из переменных синхронизации, $\perp_S^S \sqsubseteq_S^S ls \sqsubseteq_S^S \top_S^S$ и $ls \supseteq ls' \Longrightarrow ls \sqsubseteq_S^S ls'$ для всех элементов $ls, ls' \subseteq S$. $E_S^T = \{\perp_S^T, \varepsilon, \top_S^T\} \cup G$.

2. Для этого анализа множество точности содержит только один элемент: $\Pi_S = \{\emptyset\}$.

3. Оператор перехода добавляет захватываемую блокировку во множество $ls$ по время оператора *acquire* и удаляет ее из него во время оператора *release*. Формально, переход $e \rightsquigarrow_P e', e = (ls, g), e' = (ls', g'), g = (\cdot, op, \cdot)$ существует, если

- op = acquire(s) и $s \notin ls \wedge ls' = ls \cup \{s\}, g' \in G$.
- op = release(s) и $ls' = ls \backslash \{s\}, g' \in G$.
- op = thread create($l_v$) и $ls' = \emptyset, g' \in G$
- иначе, $ls' = ls, g' \in G$.

4. Оператор слияния *merge* не объединяет элементы: $merge_S(e, e', \pi) = e'$.

5. Оператор останова *stop* проверяет, существует ли состояния, которое содержит меньше захваченных блокировок: $stop_S(e, R, \pi) = \exists e' \in R : (e \sqsubseteq e')$.

6. Точность не регулируется: $prec_S(e, \pi, R) = (e, \pi)$.

7. Определение проекции: $\forall e \in E_S, e = (s, g)$: $e|_P = (s, \varepsilon)$.

Переходы в окружении ($\varepsilon$-переходы) не могут изменить множество захваченных блокировок, так как один поток не может повлиять на захваченные блокировки другого потока. Тем не менее, проекция сильно влияет на совместность состояний, так как потоки не могут захватить одну и ту же блокировку одновременно.

8. $\forall e, e' \in E_S, e = (ls, q), e' = (ls', q')$: $compose_S(e, e') = \tilde{e} = (ls, q')$.

9. $\forall e_1, e_2 \in E_S, e_i = (ls_i, q_i), ls_i \in E_S^S$: $compatible_P(e_1, e_2) = (ls \cap ls' = \emptyset)$. Проверка совместности сильно похожа на классический алгоритм Lockset. Если существует одна и та же блокировка в обоих состояниях, операции не могут быть объединены, так как два потока не могут дважды захватить одну и ту же блокировку.

## *12. CompositeCPA*

Анализ используется для комбинации различных техник анализа вместе. Примеры таких анализов были описаны выше. $\mathbb{C}$ = ($D_C$, $\Pi_C$, $\rightsquigarrow_C$, $merge_C$, $stop_C$, $prec_C$, $compatible_C$, $\cdot|_C$, $compose_C$) включает в себя n вложенных анализов $\Delta_i$ = ($D_i$, $\Pi_i$, $\rightsquigarrow_i$, $merge_i$, $stop_i$, $prec_i$, $compatible_i$, $\cdot|_i$, $compose_i$).

1. Абстрактный домен $D_C = D_1 \times \cdots \times D_n$ реализует декартово произведение всех вложенных абстрактных доменов.

2. Множество точности также реализует декартово произведение вложенных множеств точности: $\Pi_C = \Pi_1 \times \cdots \times \Pi_n$.

3. Оператор перехода применяет вложенные операторы перехода к соответствующим частям состояния. Таким образом, переход возможен, если для всех вложенных анализов возможен переход между соответствующими вложенными состояниями:

$$\forall e_1, e_2 \in E_{\mathrm{C}}: e_1 \rightsquigarrow_{\mathrm{C}} e_2 \Leftrightarrow \forall 1 \le i \le n: e_1^i \rightsquigarrow_i e_2^i$$

4. Оператор слияния *merge* вызывает вложенные операторы: $merge_{\mathrm{C}}(e_1, e_2, \pi) = (merge_1(e_1^1, e_2^1, \pi^1), \ldots, merge_n(e_1^n, e_2^n, \pi^n))$.

5. Оператор останова *stop* вызывает вложенные операторы: $\forall e \in E, R \subseteq E, \pi \in \Pi: stop_{\mathrm{C}}(e, R, \pi) = \exists \hat{e} \in R, \forall 1 \le i \le n : stop_i(e^i, \{\widehat{e^i}\}, \pi^i)$.

6. Аналогично оператор настройки точности вызывает вложенные операторы: $prec_{\mathrm{C}}(e, \pi, R) = (prec_1(e^1, \pi^1, R), \ldots, prec_n(e^n, \pi^n, R))$.

7. Аналогично оператор вычисления проекции вызывает вложенные операторы: $\forall e \in E_{\mathrm{C}}, : e|_P = (e^1|_P, \ldots, e^n|_P)$.

8. $\forall e_1, e_2 \in E_{\mathrm{C}} : compose_{\mathrm{C}}(e_1, e_2) = (compose_1(e_1^1, e_2^1), \ldots, compose_n(e_1^n, e_2^n))$.

9. $\forall e_1, e_2 \in E_{\mathrm{C}} : compatible_{\mathrm{C}}(e_1, e_2) = (compatible_1(e_1^1, e_2^1), \ldots, compatible_n(e_1^n, e_2^n))$.

## *13. Поиск состояний гонки*

Как уже было описано, поиск состояний гонки разбивается на два этапа: построение множества достижимых состояний и поиск пар, которые образуют состояние гонки. Для решения первой подзадачи применяется набор из тех CPA, которые были описаны в предыдущих разделах.

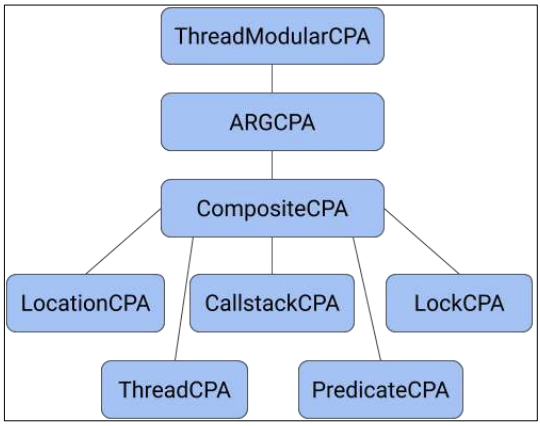


*Рис. 9. Пример конфигурации CPA*
*Fig. 9. An example of CPA configuration*

На рис. 9 представлен пример конфигурации инструмента. Так, набор CPA содержит ThreadModularCPA (раздел 6), в качестве верхнеуровневого. В него вложен ARGCPA, который обеспечивает связь между различными абстрактными состояниями, в том числе связи «родительский переход – дочерний переход», «проецируемый переход – проекция» и др. CompositeCPA (разд. 12) обеспечивает параллельную работу вложенных в него CPA: LockationCPA (разд. 7) моделирует счетчик команд потока, CallstackCPA обеспечивает межпроцедурность анализа, LockCPA (разд. 11) отслеживает захваты блокировок, ThreadCPA(разд. 8) отслеживает точки создания потоков, PredicateCPA(разд. 10) реализует анализ предикатов. Важно отметить, что это не единственная возможная конфигурация инструмента. Более того, для решения различных задач могут потребоваться различные конфигурации, то есть наборы, используемых техник анализа.

Итак, с помощью некоторой конфигурации инструмента производится построение абстрактного графа достижимости (англ. Abstract Reachability Graph, ARG) из абстрактных переходов. То есть, по завершению этого этапа имеется граф из переходов программы, которые являются достижимыми при заданном уровне абстракции, то есть, совсем не обязательно они будут достижимыми при реальном выполнении программы.

Следующим этапом необходимо найти среди этих переходов те, которые образуют состояние гонки.

Обычно под состоянием гонки подразумевается ситуация, при которой имеет место одновременный доступ к некоторой памяти из разных потоков, причем один из доступов должен быть записью. Два основных вопроса, которые возникают при статическом поиске гонок: как определить, что доступ производится к одной и той же памяти, и как определить одновременность доступов. Дальше будут подробно представлены эти две особенности в предлагаемом подходе.

При описании формальной модели программы использовался простой императивный язык программирования, который поддерживает разделяемые данные, представленные только глобальными переменными. В реальном программном обеспечении используется большое количество операций с указателями, структурами и более сложными типами данных. Для работы с ними используется модель памяти BnB, которая разделяет всю память на непересекающиеся множества регионов. Каждый регион относится к одному типу данных или полю структуры, в случае если у него не брался адрес. Такая модель памяти имеет ряд ограничений. В первую очередь, она не полностью поддерживает адресную арифметику и кастирование, что накладывает некоторые дополнительные условия на применимость подхода. Кроме того, она может приводить к ложным предупреждениям об ошибках, в случае если два указателя одного типа никогда не указывают на одну и ту же память.

В случае, если обнаружилась пара доступов к одной области памяти, в данном случае, к одному региону, необходимо проверить возможность одновременного доступа к ней. Для этого используется понятие совместности переходов. Совместность означает, что два частичных абстрактных состояния могут быть частью одного глобального состояния. Или, другими словами, один переход может быть применен, как переход в окружении, к другому переходу и наоборот, откуда следует, что эти два перехода могут быть выполнены параллельно. Таким образом, предложенный подход является обобщением подхода Lockset[13], который определяет состояние гонки, как два доступа к некоторой памяти с непересекающимся множеством блокировок. Одним из ограничений подхода Lockset является отсутствие поддержки других типов синхронизации. В расширении подхода для этого используется оператор compatible. Так как проверка совместности использует различные типы анализа, включая анализ примитивов синхронизации, анализ предикатов и другие, такой способ является более точным, чем алгоритм Lockset.

Итак, алгоритм поиска состояний гонки состоит из следующих шагов:

1. построить множество достижимых абстрактных переходов (алгоритм 1);

2. для каждого перехода определить регионы памяти, к которым может производиться доступ;

3. для каждого региона памяти попытаться подобрать пару совместных переходов, которые образуют состояние гонки;

4. проверить каждую пару путей, приводящих к состоянию гонки, на достижимость и уточнить предикатную абстракцию в случае необходимости [21].

Следует добавить, что алгоритм уточнения абстракции по контрпримерам (англ. Counterexample Guided Abstraction Refinement, CEGAR [11]) был использован без существенных модификаций. Однако, в таком виде он позволяет проводить уточнение только локальных путей в потоке, то есть не позволяет обнаружить противоречие между различными потоками. Однако, это не является принципиальным ограничением подхода, и при соответствующем расширении метода CEGAR на случай подхода с раздельным рассмотрением потоков, возможно получение более точных результатов.

## *14. Результаты*

Подход с раздельным помодульным анализом потоков и абстракными переходами был реализован в инструменте CPAchecker, как композиция следующих типов анализа:

1. Анализ точек программы (англ. Location analysis) L;
2. Анализ стека вызовов функци (англ. Callstack analysis) CS;
3. Анализ потоков (англ. Thread Analysis) T;
4. Анализ примитивов синхронизации (англ. Lock analysis) S (раздел 8);
5. Анализ предикатов с опциями (англ. Predicate analysis) P (раздел 7):
   I. Join – объединение абстрактных состояний и абстрактных дуг. В данном варианте все эффекты окружения предикатного анализа объединяются в один, что, естественно, снижает точность, но повышает скорость.
   II. Eq – объединение абстрактных дуг, если равны абстрактные состояния;
   III. Sep – переходы никогда не объединяются.
6. Анализ явных значений (Value analysis) V [20].

### 14.1 Результаты на наборе задач SV-COMP

Набор задач SV-COMP состоит из 1082 задач, большая часть из которых являются небольшими примерами около 100 строк кода. Однако, они содержат в себе нетривиальные механизмы синхронизации, в том числе алгоритмы Деккера, Петерсона и др. 7 задач были подготовлены на основе драйверов ОС Linux. Все задачи доступны в официальном репозитории SV-COMP[4]. Эксперименты проводились с использованием кластера из 191 машины VerifierCloud[5]. Были использованы ограничения по памяти в 8 Гб и по времени 15 минут.

Оценка результатов реализованного подхода в инструменте CPAchecker была проведена для следующих вариантов анализа:

1. Подходы с раздельным анализом потоков с абстракцией переходов:
   a. Варианты объединения для предикатного анализа
      i. *MergeJoin*. (L,CS,T,S,P) Join.
      ii. *MergeEq*. (L,CS,T,S,P) Eq.
      iii. *MergeSep*. (L,CS,T,S,P) Sep.
   b. *Value*. Анализ явных значений (L, CS, T, S, V).
2. *Threading*. Анализ чередований, описанный в [17] использует классическую теорию CPA и рассматривает все возможные чередования потоков.

*Табл. 1. Результаты экспериментов*
*Table 1. Experiment Results*

| Подход | MergeJoin | MergeEq | MergeSep | Value | Threading |
|---|---|---|---|---|---|
| Найденные ошибки | 1026 | 1027 | 763 | 963 | 720 |
| Истинные | 805 | 806 | 548 | 752 | 720 |
| Ложные | 221 | 221 | 215 | 211 | 0 |
| Доказательства корректности | 27 | 28 | 28 | 30 | 165 |
| Истинные | 27 | 28 | 28 | 30 | 165 |
| Ложные | 0 | 0 | 0 | 0 | 0 |

[4] https://github.com/sosy-lab/sv-benchmarks
[5] https://vcloud.sosy-lab.org/cpachecker/webclient/master/info

| | | | | | |
|---|---|---|---|---|---|
| Незавершенный анализ | 29 | 27 | 297 | 89 | 197 |
| Процессорное время, с | 16 800 | 15 900 | 278 000 | 75 300 | 93 700 |
| Астрономическое время, с | 10 200 | 9 630 | 258 000 | 64 900 | 67 500 |

Результаты (табл. 1) подтверждают, что подход с раздельным анализом потоков не пропускает ошибок при некоторых заранее известных ограничениях.

*MergeJoin* показывает результаты лучше, чем конфигурация *MergeSep*. Это происходит, в основном, из-за большого количества переходов в окружении, которые *MergeSep* рассматривает по одному. *MergeJoin* объединяет их в один и применяет за один раз. Это позволяет сохранить огромное количество времени. В то же время *MergeSep* позволяет избежать некоторых неточностей из-за анализа переходов по-отдельности и выдает меньшее количество ложных сообщений об ошибках.

Конфигурация *Value* является достаточно простым и быстрым анализом, но иногда она вынуждена рассматривать все возможные варианты значений переменных, что приводит к исчерпанию ресурсов по времени.

Классический анализ *Threading* является корректным и точным и не выдает ложных вердиктов. Однако, он требует значительного числа ресурсов, это является главным недостатком подхода. Эта конфигурация решила только один из семи сложных задач, основанных на драйверах операционной системы Linux. Подходы с анализом потоков по-отдельности (*MergeJoin*, *MergeEq*, *MergeSep*) решают пять из семи таких задач.

Большая часть новых доказательств корректности, полученных новыми подходами, (26 из 27 для *MergeJoin*) не находились классическими методами (*Threading*). Это также является одним из важных вкладов данного метода.

## 14.2 Поиск состояний гонки в драйверах устройств

Верификационные задачи, основанные на драйверах операционной системы Linux, были подготовлены системой Klever, которая предназначена для верификации различного программного обеспечения [17, 18]. Она разделяет большой объем целевого исходного кода на отдельные небольшие верификационные задачи. Для ядра операционной системы Linux верификационная задача соответствует одному модулю. Система Klever автоматически готовит модель окружения модуля, которая включает в себя модель потоков, модель сердцевины ядра и операций над модулем. После подготовки верификационной задачи Klever запускает верификацию через общий интерфейс – BenchExec [23].

Сравнение проводилось на подсистеме drivers/net/ ядра операционной системы Linux 4.2.6, для которой Klever подготовил 425 верификационных задач. Эксперименты также проводились с использованием кластера из 191 машины VerifierCloud. Были использованы ограничения по памяти в 8 Гб и по времени 15 минут.

Алгоритм поиска состояний гонки с помощью инструмента был описан в разд. 12. При этом подготовленные задачи не содержали кодирование состояния гонки в виде проверки задачи достижимости, поэтому было невозможно снова включить в сравнение участников соревнования SV-COMP, так как они не поддерживают поиск состояний гонки.

В сравнении (табл. 2) участвовали следующие конфигурации:

1. *Base*. Конфигурация *MergeJoin* из предыдущего пункта.

2. *Havoc*. Конфигурация *MergeJoin* из предыдущего пункта без возможности извлечения предикатов над глобальными переменными. Это означает, что анализ не учитывает значения глобальных переменных и считает, что они могут иметь случайное значение.

*Табл. 2. Сравнение конфигураций Base и Havoc*
*Table 2. Comparing Base and Havoc Configurations*

| Подход | Base | Havoc |
|---|---|---|
| Модули с ошибкой | 6 | 26 |
| Корректные модули | 262 | 254 |
| Незавершенный анализ | 157 | 145 |
| Процессорное время, с | 137 000 | 125 000 |

Модули с ошибкой означают, что в них было найдено хотя бы одно потенциальное состояние гонки. Конфигурация *Havoc* чуть более быстрая, но менее точная, поэтому она не может доказать корректность 8 корректных модулей, однако способна обнаружить больше ошибок. При этому 6 из этих ошибок были найдены в пропущенных корректных модулях, что означает, что найденные ошибки – ложные из-за неточности анализа. Но 13 ошибок были соответствуют незавершенному анализу у *Base* конфигурации, что означает, что *Havoc* может находить новые ошибки.

Если изучить эти 8 модулей, которые были доказаны, как корректные, то окажется, что в них структура данных устройства (дескриптора устройства) была некорректно инициализирована при подготовке верификационной задачи, и поэтому в ней действительно нет состояния гонки. На самом деле проблема связана с подготовкой модели окружения модуля (часть инфраструктуры Klever), так как они не учитывает семантику данных.

Часть полученных ошибок была проанализирована. Обычно для каждой верификационной задачи могут быть получены несколько потенциальных состояний гонки. Будем называть их предупреждениями. Соотношение истинных предупреждений к общему количеству составляет около 42% (34 корректных предупреждения и 47 некорректных). Основной проблемой ложных предупреждений об ошибках являются проблемы с моделью памяти (более 90%). Остальные предупреждения связаны со спецификой ядра (например, обработкой прерываний), анализом функциональных указателей, другими примитивами синхронизации и др. Интересно, что таких проблем, связанных с неточностью подхода, как были в задачах SV-COMP, в модулях операционной системы Linux не наблюдается. Это подтверждает предположение, что задачи SV-COMP являются слишком искусственными, по крайней мере в категории многопоточных задач (Concurrency).

Из 24 сообщений о возможном состоянии гонки 14 были подтверждены разработчиками, для 8 не было получено обратного ответа, и только в двух случаях найденная гонка была признана недостижимой из-за аппаратной синхронизации устройства. Однако большинство истинных ошибок были найдены в «древних» драйверах, а их исправление не является актуальной задачей. Только 4 исправления из 14 были включены в основную ветку разработки ядра.

Результаты экспериментов позволяют утверждать, что удалось успешно интегрироваться с другими техниками анализа, которые доступны во фреймворке CPAchecker. Такие подходы, как CEGAR, могут быть применены с минимальным количеством изменений технического характера. Некоторые реализации конкретных видов анализа могут быть использованы без модификаций (AutomatonCPA) или с минимальными изменениями в коде (CallstackCPA). Для некоторых существующих техник анализа (PredicateCPA, ValueCPA, CompositeCPA) была реализована поддержка операций над эффектами окружения, при этом основная функциональность анализа осталась без изменений. Изменения параметров работы инструмента (вариантов анализа) достигается лишь опцией запуска, то есть пересобирать инструмент не требуется.

Из результатов видно, что для разных целей может быть использованы различные варианты работы инструмента: для небольших задач, типа SV-COMP, необходимы более точные подходы. Для прикладных задач, типа драйверов, могут быть полезны менее точные варианты анализа, чтобы иметь возможность получить некоторую оценку корректности. Вместе с тем, более точные подходы, могут применяться для глубокой верификации, которая требует определенных ресурсов.

## *15. Обзор похожих работ*

Существующие подходы к анализу многопоточного программного обеспечения имеют различные свойства и производительность. С одной стороны, существуют точные подходы, которые могут доказать корректность программ при некоторых определенных предположениях. Начиная с ограничиваемой проверки моделей, большинство подходов прилагают значительные усилия для оптимизации обхода пространства состояний программы. Примерами таких оптимизаций могут стать редукция частичных порядков [1], ограничение контекста [24, 25], и др.

Попыткой абстрагироваться от нерелевантного окружения является подход с раздельным анализом потоков (thread-modular approach), который впервые был предложен в [7]. Эта версия еще не использовала никакую абстракцию. Подход с раздельным анализом потоков для формальной верификации был представлен в [26]. Основная идея была в поиске инвариантов для каждого процесса, из которых потом должно следовать проверяемое свойство. Оценка подхода проводилась для двух протоколов взаимного исключения. Предикатная абстракция была добавлена к этому подходу в [27]. Основным отличием было то, что рассматривался только один поток из нескольких копий. Поэтому окружение формировалось этим же потоком. При этом никакие примитивы синхронизации не рассматривались.

Расширение подхода с раздельным анализом потоков, которое использует абстракцию, был представлен в [28] и затем реализован в инструменте TAR [8]. Описанный в данной статье подход имеет ряд отличий:

TAR рассматривает примитивы синхронизации, как обычные переменные. Предложенный метод использует специальный анализ блокировок, который может применяться совместно с другими вариантами анализа. Это позволяет избегать лишних итераций уточнения абстракции.

TAR применяет эффекты потока, которые напрямую получены из операторов программы. CPALockator предоставляет возможность абстракции (оператор $e|_P$) и объединения (оператор $merge$) различных переходов. Это может позволяет выбирать баланс между скоростью анализа и его точностью.

TAR поддерживает фиксированное количество потоков, в то время как предложенный метод теоретически поддерживает неограниченное количество потоков.

Для построения окружения TAR использует аппроксимацию снизу, а предложенный метод – аппроксимацию сверху.

Похожий подход также был реализован в инструменте Threader [29]. Этот инструмент использует аппроксимацию сверху для построения окружения, основанную на дизъюнктах Хорна. Также как и в инструменте CPALockator, Threader может искать модульные доказательства, но при этом он может искать и немодульные, в которых учитывается полной взаимодействие между потоками.

Существуют различные техники трансформации параллельной программы в последовательную (англ. sequentialization) для последующей проверки ее обычными инструментами статической верификации [30–32]. Один из примеров является WHOOP [33], который использует эту технику трансформации и не учитывает взаимодействие потоков. Более того, он сильно использует алгоритм Lockset и не может быть расширен

какими-нибудь другими анализами. Он применяется в качестве преданализа для более точного инструмента CORALL [34].

С другой стороны, существуют множество легковесных подходов, которые могут применяться к огромному объему кода. Такие техники определяются слабыми требованиями к ресурсам и низкой точностью. Примерами являются RELAY [35] и Locksmith [36], которые применялись для анализа кода операционной системы Linux. RELAY нашел несколько тысяч предупреждений, что потребовало применения неточных фильтров для сокращения этого количества. Этот инструмент вообще не учитывает взаимодействие потоков.

Инструмент Locksmith, напротив, учитывает точки создания потоков, но не может точно определить разделяемые данные и способы взаимодействия потоков. Он вычисляет общий эффект от потока. В терминах представленной теории используется оператор $merge$, объединяющий все переходы в окружении в один. При этом экспериментальная оценка Locksmith показала, что он имеет проблемы с масштабируемостью.

В [37] авторы представили расширение анализа алиасов Андерсена на случай многопоточных программ. Идея был похожа на подход с раздельным анализом потоков, так как вычислялось множество операторов, которое могло выполняться параллельно, а затем эти операторы применялись к другим потокам.

Также существуют различные подходы для поиска состояний гонки в специфичном программном обеспечении. Например, в низкоуровневом программном обеспечении со вложенными прерываниями [38], поиск гонок во FreeRTOS [39, 40]. А также состояний гонки специального вида – использования памяти после ее освобождения (англ. use-after-free) в драйверах устройств операционной системы Linux. Такие подходы показывают достаточно хорошие результаты, но значительно используют специфику исходного кода или проверяемого свойства. Представленный в данной статье теоретический подход претендует на то, чтобы быть более общим, однако это не отменяет того факта, что он может быть также улучшен при использовании дополнительной информации об исходном коде или проверяемом свойстве.

## *16. Заключение*

В работе представлен подход к практическому поиску состояний гонки в сложном программном обеспечении. Была расширена теория CPA и реализована в новом инструменте. Эксперименты показали преимущества подхода на больших верификационных задачах перед существующими техниками статической верификации. Небольшие задачи со сложным взаимодействием потоков лучше решаются другими инструментами, так как предложенный подход абстрагируется от такого взаимодействия. Однако наш подход также не пропускает ошибок (при некоторых предположениях) и может быть развит в будущем.

Таким образом, можно заключить, что основные требования к новому инструменту были выполнены, так как он успешно применяется к различным программным системам, в том числе, к драйверам ОС Linux. Так же, как и в классических методах статической верификации, может быть получена гарантия отсутствия ошибок при выполнении указанных предположений: требований на операторы CPA и условий корректного разбиения на непересекающиеся регионы BnB модели.

Предложенная теория позволяет описывать достаточно сложные варианты анализа, в том числе и те, которые не включаются в понятие анализа с раздельным рассмотрением потоков. Тем не менее, эта теория не исчерпывает всех возможных подходов, и для эффективного описания масштабируемого подхода с частичным вычислением чередований потребуется ее расширение. Однако, эта тема выходит за рамки данной статьи.

Одним из возможных направлений развития подхода является добавление взаимодействия потоков, возможно, не в полном объеме, чтобы сохранить масштабируемость. Это может быть отдельный анализ CPA, который будет динамически настраивать свою точность с помощью алгоритма CEGAR, обеспечивая некоторый промежуточный вариант между подходом с раздельным анализом потоков и перебором чередований.

Другим возможным улучшением инструмента может стать интеграция его с другим подходом. Например, комбинация быстрого подхода с раздельным анализом потоков в качестве первого этапа, а затем классический тяжеловесный анализ – на втором этапе. Это может быть реализовано в соответствии с идеей кооперативной верификации [42].

Еще одним возможным направлением развития подхода является построение реального пути с чередованием потоков на основе пути с примененными эффектами окружения. Этот путь был бы полезен для исследования и уточнения абстракции.

## Список литературы / References


[1]. Parosh Abdulla, Stavros Aronis, Bengt Jonsson, and Konstantinos Sagonas. Optimal dynamic partial order reduction. SIGPLAN Notices, vol. 49, issue 1, 2014, pp. 373–384.

[2]. Patrice Godefroid. Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem. Springer-Verlag, Berlin, Heidelberg, 1996, 143 p.

[3]. Gérard Basler, Michele Mazzucchi, Thomas Wahl, and Daniel Kroening. Symbolic counter abstraction for concurrent software. Lecture Notes in Computer Science, vol. 5643, 2009, pp. 64–78.

[4]. Dirk Beyer. Automatic verification of C and Java Programs: SV-COMP. Lecture Notes in Computer Science, vol. 11429, 2019, pp. 133–155.

[5]. Dirk Beyer, Thomas A. Henzinger, and Grégory Théoduloz. Configurable software verification: concretizing the convergence of model checking and program analysis. Lecture Notes in Computer Science, vol. 4590, 2007, pp. 504–518

[6]. D. Beyer, T.A. Henzinger, and G. Theoduloz. Program analysis with dynamic precision adjustment. In Proc. of the 23rd IEEE/ACM International Conference on Automated Software Engineering, 2008, pp. 29–38.

[7]. Cormac Flanagan and Shaz Qadeer. Thread-modular model checking. Lecture Notes in Computer Science, vol. 2648, 2003, pp. 213–224.

[8]. Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Shaz Qadeer. Thread-Modular Abstraction Lecture Notes in Computer Science, vol. 2725, 2003, pp, 262–274.

[9]. Byron Cook, Daniel Kroening, and Natasha Sharygina. Verification of Boolean programs with unbounded thread creation. Theoretical Computer Science, vol. 388, issue 1-3, 2007, pp. 227–242.

[10]. Ashutosh Gupta, Corneliu Popeea, and Andrey Rybalchenko. Threader: A constraint-based verifier for multi-threaded programs. Lecture Notes in Computer Science, vol. 6806, 2011, pp. 412–417.

[11]. E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. Lecture Notes in Computer Science, vol. 1855, 2000, pp. 154–169.

[12]. Susanne Graf and Hassen Saidi. Construction of abstract state graphs with PVS. Lecture Notes in Computer Science, vol. 1254, 1997, pp. 72–83.

[13]. Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, and Thomas Anderson. Eraser: A dynamic data race detector for multi-threaded programs. ACM SIGOPS Operating Systems Review, vol. 31, issue 5, 1997, pp. 27–37.

[14]. Richard Bornat. Proving pointer programs in Hoare logic. Lecture Notes in Computer Science, vol. 1837, 2000, pp. 102–126.

[15]. R M Burstall. Some techniques for proving correctness of programs which alter data structures. Machine Intelligence, vol. 7, 1972, pp. 23–50.

[16]. Pavel Andrianov, Karlheinz Friedberger, Mikhail Mandrykin, Vadim Mutilin, and Anton Volkov. CPA-BAM-BnB: Block-abstraction memoization and region-based memory models for predicate abstractions. Lecture Notes in Computer Science, vol. 10206, 2017, pp. 355–359.

[17]. Evgeny Novikov and Ilja Zakharov. Towards automated static verification of GNU C programs. Lecture Notes in Computer Science, vol. 10742, 2018, pp. 402–416.

[18]. Evgeny Novikov and Ilja Zakharov. Verification of operating system monolithic kernels without extensions. Lecture Notes in Computer Science, vol. 11247, 2018, pp. 230–248.
[19]. Dirk Beyer and Karlheinz Friedberger. In Proc. of the 11th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, 2016, pp. 61–71.
[20]. Dirk Beyer and Stefan Löwe. Explicit-state software model checking based on CEGAR and interpolation. Lecture Notes in Computer Science, vol. 7793, 2013, pp. 146–162.
[21]. D. Beyer, M.E. Keremoglu, and P. Wendler. Predicate abstraction with adjustable-block encoding. In Proc. of 10th International Conference on Formal Methods in Computer-Aided Design, 2010, pp. 189–197.
[22]. Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. Lecture Notes in Computer Science, vol. 1579, 1999, pp. 193–207
[23]. Dirk Beyer, Stefan Löwe, and Philipp Wendler. Reliable benchmarking: requirements and solutions. International Journal on Software Tools for Technology Transfer, vol. 21, issue 1, 2019, pp. 1–29.
[24]. Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. Lecture Notes in Computer Science, vol. 3440, 2005, pp. 93–107.
[25]. Lucas Cordeiro, Jeremy Morse, Denis Nicole, and Bernd Fischer. Context-bounded model checking with esbmc 1.17. Lecture Notes in Computer Science, vol. 7214, 2012, pp. 534–537.
[26]. Ariel Cohen and Kedar S. Namjoshi. Local proofs for global safety properties. Formal Methods in System Design, vol. 34, issue 2, 2009, pp. 104–125.
[27]. Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. Race checking by context inference. In Proc. of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation, 2004, pp. 1–13.
[28]. Alexander Malkis, Andreas Podelski, and Andrey Rybalchenko. Thread-modular verification is cartesian abstract interpretation. Lecture Notes in Computer Science, vol. 4281, 2006, pp. 183–197.
[29]. Ashutosh Gupta, Corneliu Popeea, and Andrey Rybalchenko. Predicate abstraction and refinement for verifying multi-threaded programs. ACM SIGPLAN Notices, vol. 46, issue 1m 2011, pp. 331–344.
[30]. Akash Lal and Thomas Reps. Reducing concurrent analysis under a context bound tosequential analysis. Formal Methods in System Design, vol. 35, issue 1, 2009, pp. 73–97.
[31]. Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Reducing context-bounded concurrent reachability to sequential reachability. Lecture Notes in Computer Science, vol. 5643, 2009, pp. 477–492.
[32]. Ermenegildo Tomasco, Omar Inverso, Bernd Fischer, Salvatore La Torre, and Gennaro Parlato. MU-CSeq: Sequentialization of C programs by shared memory unwindings. Lecture Notes in Computer Science, vol. 8413, 2014, pp. 402–404.
[33]. Pantazis Deligiannis, Alastair F. Donaldson, and Zvonimir Rakamaric. Fast and precise symbolic analysis of concurrency bugs in device drivers (t). In Proc. of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015, pp. 166–177.
[34]. Akash Lal, Shaz Qadeer, and Shuvendu K. Lahiri. A solver for reachability modulo theories. Lecture Notes in Computer Science, vol. 7358, 2012, pp. 427–443.
[35]. Jan Wen Voung, Ranjit Jhala, and Sorin Lerner. RELAY: Static race detection on millions of lines of code. In Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, 2007, pp. 205–214.
[36]. Polyvios Pratikakis, Jeffrey S. Foster, and Michael Hicks. LOCKSMITH: Context-sensitive correlation analysis for race detection. In Proc. of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2006, pp. 320–331.
[37]. Peng Di and Yulei Sui. Accelerating dynamic data race detection using static thread interference analysis. In Proc. of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores, 2016, pp. 30–39.
[38]. Daniel Kroening, Lihao Liang, Tom Melham, Peter Schrammel, and Michael Tautschnig. Effective verification of low-level software with nested interrupts. In the Europe Conference & Exhibition on Design, Automation & Test, 2015, pp. 229–234.
[39]. Suvam Mukherjee, Arun Kumar, and Deepak D'Souza. Detecting all high-level dataraces in an RTOS kernel. Lecture Notes in Computer Science, vol. 10145, 2017, pp. 405–423.
[40]. Nikita Chopra, Rekha Pai, and Deepak D'Souza. Data races and static analysis for interrupt-driven kernels. Lecture Notes in Computer Science, vol. 11423, 2019, pp. 697–723.

[41]. Jia-Ju Bai, Julia Lawall, Qiu-Liang Chen, and Shi-Min Hu. Effective static analysis of concurrency use-after-free bugs in Linux device drivers. In the USENIX Annual Technical Conference, 2019, pp. 255–268.

[42]. Dirk Beyer, Marie-Christine Jakobs, Thomas Lemberger, and Heike Wehrheim. Reducer-based construction of conditional verifiers. In Proceedings of the 40th International Conference on Software Engineering, 2018, pp. 1182–1193.

## Информация об авторах / Information about authors

Павел Сергеевич АНДРИАНОВ – младший научный сотрудник. Сфера научных интересов: статическая верификация, анализ многопоточных программ.

Pavel Sergeevich ANDRIANOV – junior researcher. Research interests: software model checking, analysis of multithreaded software.

**DOI: 10.15514/ISPRAS-2019-31(5)-17**



# Процедуры поиска лорановых и регулярных решений линейных дифференциальных уравнений с усеченными степенными рядами в роли коэффициентов

*С.А. Абрамов, ORCID: 0000-0001-7745-5132 <sergeyabramov@mail.ru>*
*Д.Е. Хмельнов, ORCID: 0000-0002-4602-2382 <dennis_khmelnov@mail.ru>*
*А.А. Рябенко, ORCID: 0000-0001-5780-7743 <anna.ryabenko@gmail.com>*

*Вычислительный центр им. А.А. Дородницына*
*Федеральный исследовательский центр «Информатика и управление» РАН,*
*119333, Россия, Москва, ул. Вавилова, 40*

**Аннотация.** Обсуждаются задачи построения лорановых и регулярных решений линейных обыкновенных дифференциальных уравнений. Предполагается, что коэффициентами уравнений являются формальные степенные ряды, которые заданы в виде их усечений, то есть в виде начальных отрезков рядов, и что степень этих начальных отрезков может быть различна. Рассматриваемые виды решений также содержат степенные ряды. Интересует нахождение максимально возможного числа коэффициентов этих рядов в решениях, таких что они являются инвариантными относительно различных возможных продлений усечений рядов – коэффициентов заданного уравнения. В настоящей статье дается беглый обзор алгоритмов для решения такой задачи и представляется реализация этих алгоритмов в виде Maple-процедур. Рассматриваются линейные обыкновенные дифференциальные уравнения с бесконечными (формальными) степенными рядами в роли коэффициентов, при этом эти ряды задаются в усеченном виде. Предлагаются компьютерно-алгебраические процедуры (они реализованы в среде Maple) построения решений двух видов. Эти решения содержат, в свою очередь, степенные ряды. Исходя из заданных усеченных рядов-коэффициентов уравнения, процедуры находят максимально возможное число членов рядов, входящих в решения.

**Ключевые слова:** усеченные степенные ряды; линейные обыкновенные дифференциальные уравнения; лорановы решения; регулярные решения; компьютерная алгебра; Maple

**Для цитирования:** Абрамов С.А., Рябенко А.А., Хмельнов Д.Е Процедуры поиска лорановых и регулярных решений линейных дифференциальных уравнений с усеченными степенными рядами в роли коэффициентов. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 233-248. DOI: 10.15514/ISPRAS-2019-31(5)-17

**Благодарности:** Исследование частично поддержано РФФИ, грант 19-01-00032

# Procedures to search for Laurent and regular solutions of linear ordinary differential equations with truncated power series coefficients

*S.A. Abramov, ORCID: 0000-0001-7745-5132 <sergeyabramov@mail.ru>*
*D.E. Khmelnov, ORCID: 0000-0002-4602-2382 <dennis_khmelnov@mail.ru>*
*A.A. Ryabenko, ORCID: 0000-0001-5780-7743 <anna.ryabenko@gmail.com>*

*Dorodnicyn Computing Center,*
*Federal Research Center «Computer Science and Control» of Russian Academy of Sciences,*
*40, ul. Vavilova, Moscow, 119333, Russia*

**Abstract.** We previously published algorithms for searching the so-called Laurent and regular solutions of linear ordinary differential equations with infinite formal power series in the role of coefficients. The question of infinite series representation is very important for computer algebra. In those algorithms the series are given in truncated form, which means that we do not have complete information about the equation under consideration. Based on this incomplete information, algorithms give the maximum possible number of terms of the series included in the solutions. We are interested in the information about these solutions that is invariant to possible prolongations of those truncated series that represent the coefficients of the equation. The mentioned publications reported preliminary (trial) versions for procedures, which implement these algorithms, as well as experiments with them. To date, the procedures have been improved, the interface and data presentation are designed for them in a uniform manner. The advanced procedures are discussed in the current paper. The various examples are presented which illustrates the use of the procedures, including their optional parameters. These procedures are available from the web page http://www.ccas.ru/ca/TruncatedSeries.

Keywords: truncated power series; linear ordinary differential equations; Laurent solutions; regular solutions; computer algebra; Maple

**For citation:** Abramov S.A., Khmelnov D.E., Ryabenko A.A. Procedures to search for Laurent and regular solutions of linear ordinary differential equations with truncated power series coefficients. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019, pp. 233-248 (in Russian). DOI: 10.15514/ISPRAS-2019-31(5)-17

**Acknowledgements.** The study was partially supported by RFBR, grant 19-01-00032

## *1. Введение*

В [1], [2] были предложены алгоритмы поиска так называемых лорановых и регулярных решений (определения даются в подразд. 2.2) линейных обыкновенных дифференциальных уравнений с бесконечными формальными степенными рядами в роли коэффициентов.

Вопрос представления бесконечных рядов очень существен для компьютерной алгебры. В данном случае ряды задаются в усеченном виде, что означает, что мы не располагаем полной информацией об уравнении. Исходя из этой неполной информации, алгоритмы дают максимально возможное число членов рядов, входящих в решения.

В [1], [2] сообщалось о предварительных (пробных) вариантах реализующих эти алгоритмы процедур и экспериментах с ними. К настоящему времени процедуры усовершенствованы, интерфейс и представление данных выбраны для них единообразно. Усовершенствованные процедуры обсуждаются ниже в статье. Эти процедуры находятся в свободном доступе и доступны на веб-станице http://www.ccas.ru/ca/TruncatedSeries.

## *2. Предварительные сведения*

### 2.1 Уравнения, операторы, усеченные ряды

Пусть $K$ – алгебраически замкнутое числовое поле. Для кольца полиномов от $x$ над $K$ мы в дальнейшем используем обычные обозначение $K[x]$. Кольцо формальных степенных рядов от $x$ над $K$ обозначается через $K[[x]]$, поле формальных лорановых рядов – через $K((x))$. Для ненулевого элемента $a(x) = \sum a_i\, x^i$, принадлежащего $K((x))$, его *валюация* $val_x a(x)$ определена равенством $val_x a(x) = min\{i | a_i \neq 0\}$, при этом $val_x 0 = \infty$. Пусть $t \in \mathbb{Z} \cup \{-\infty\}$. *$t$-усечение* $a^{\langle t \rangle}(x)$ получается отбрасыванием всех членов $a(x)$ степени большей, чем $t$; если $t = -\infty$, то $a^{\langle t \rangle}(x) = 0$. Число $t$ называется *степенью усечения*.

В настоящей статье дифференциальные уравнения будет удобно записывать с помощью операции $\theta = x\frac{d}{dx}$ вместо обычной операции дифференцирования (переход от одной формы записи к другой выполняется легко). Мы рассматриваем уравнения вида

$$a_r(x)\theta^r y + a_{r-1}(x)\theta^{r-1} y + \cdots + a_0(x) y = 0, \tag{1}$$

где $y$ – неизвестная функция $x$. Относительно $a_0(x), a_1(x), \ldots, a_r(x)$ (будем называть их *коэффициентами уравнения*) предполагается, что $a_i(x) \in K\big[[x]\big]$, при этом *ведущий* коэффициент $a_r(x)$ не равен нулю. Также предполагается, что валюация хотя бы одного из коэффициентов $a_0(x), a_1(x), \ldots, a_r(x)$ равна нулю.

Уравнение (1) может быть записано как $\mathcal{L}(y) = 0$, где оператор $\mathcal{L}$ имеет вид

$$\sum_{i=0}^{r} a_i(x)\, \theta^i \in K\big[[x]\big][\theta], \tag{2}$$

число $r$ является *порядком* оператора $\mathcal{L}$.

Далее считаем, что задан дифференциальный оператор $L$ с полиномиальными коэффициентами:

$$\sum_{i=0}^{r} a_i(x)\, \theta^i \in K[x][\theta], \tag{3}$$

и заданы неотрицательные целые $t_o, t_1, \ldots, t_r$ такие, что $t_i \geq \deg a_i(x), i = 0,1, \ldots, r$. Предполагается при этом, что $a_r(x) \neq 0$ и что валюация по крайней мере одного из полиномов $a_0(x), a_1(x), \ldots, a_r(x)$ равна нулю:

$$\exists i\colon \mathrm{val}\; a_i(x) = 0 \tag{4}$$

**Определение 1.** *Продолжением* оператора $L$ будем называть любой оператор

$$\tilde{L} = \sum_{i=0}^{r} \tilde{a}_i(x)\, \theta^i \in K\big[[x]\big][\theta],$$

для которого $\tilde{a}_i(x) - a_i(x) = O(x^{t_i+1})$, т.е.$\mathrm{val}\left(\tilde{a}_i(x) - a_i(x)\right) > t_i, i = 0.1, \ldots, r$.

*Усеченному дифференциальному уравнению*

$$\sum_{i=0}^{r} (a_i(x) + O(x^{t_i+1}))\theta^i y = 0, \tag{5}$$

$t_i \geq \deg a_i(x), i = 0,1, \ldots, r$, мы сопоставляем оператор (3), а также набор чисел $t_0, t_1, \ldots, t_r$. Продолжение оператора (3) будет в этом случае называться также *продолжением уравнения*.

Если $L$ (или $\mathcal{L}$) – некоторый дифференциальный оператор, то под *решениями оператора* $L$ (или $\mathcal{L}$) мы понимаем решения уравнения $L(y) = 0$ (соответственно, $\mathcal{L}(y) = 0$).

В случае, когда $L$ – усеченный вариант оператора $\mathcal{L}$, будем называть $L$ и $L(y) = 0$ *усечениями* оператора $\mathcal{L}$ и соответственно уравнения $\mathcal{L}(y) = 0$.

## 2.2 Лорановы и регулярные решения уравнений

Решение дифференциального уравнения, являющееся формальным лорановым рядом, называется *лорановым*.

*Регулярное* решение имеет вид

$$y(x) = x^{\lambda} w(x), \tag{6}$$

где $\lambda \in K, w(x) \in K\big((x)\big)[\ln x]$. Каждое такое решение записывается как

$$x^{\lambda} \sum_{s=0}^{k} g_{k-s}(x) \frac{\ln^s x}{s!}, \tag{7}$$

где $k \in \mathbb{Z}_{\geq 0}$ и $g_s(x) \in K\big((x)\big), s = 0,1, \dots, k$. Мы в этом случае говорим, что $x^{\lambda}$ – степенной множитель решения $y(x)$.

Набор

$$x^{\lambda_1}, x^{\lambda_2}, \dots, x^{\lambda_p} \tag{8}$$

называется *полным* набором степенных множителей регулярных решений уравнения $\mathcal{L}(y) = 0$, если

- среди показателей степени элементов набора (8) нет различающихся на целое число,
- каждый элемент $x^{\lambda_i}$ набора (8) является степенным множителем для некоторого ненулевого регулярного решения уравнения $\mathcal{L}(y) = 0$,
- для каждого ненулевого регулярного решения уравнения $\mathcal{L}(y) = 0$ среди (8) найдется степенной множитель для этого решения.

**Примечание 1.** Согласно принятому в компьютерной алгебре определению (см., например, [3]) любая линейная комбинация над $K$ решений вида (6) также называется регулярным решением.

# *3. Алгоритмы построения решений*

## 3.1 Лорановы решения

Пусть дифференциальное уравнение $\mathcal{L}(y) = 0$ имеет ненулевые лорановы решения и $y(x) = \sum_{n=v}^{\infty} c_n x^n$ – общее лораново решение, его коэффициенты $c_n$ содержат произвольные постоянные. Алгоритм, предложенный в [4, разд. 6], позволяет для $\mathcal{L}(y) = 0$ построить усечение общего лоранового решения любой заданной степени $m$: $y^{\langle m \rangle}(x) = \sum_{n=v}^{m} c_n x^n$.

В [1] показано, что для уравнения (5) с усеченными коэффициентами можно найти усечения максимально высоких степеней для лорановых решений, обеспечивая при этом инвариантность эти усечений относительно всевозможных продолжений уравнения. Был предложен алгоритм, входом которого являются оператор $L \in K[x][\theta]$ и неотрицательные целые $t_0, t_1, \dots, t_r$, имеющие тот же смысл, что и в (3). Алгоритм строит конечное множество $W$ выражений вида

$$y^{\langle m \rangle}(x, C_1, \dots, C_r) + O(x^{m+1}), \tag{9}$$

где $y(x, C_1, \dots, C_r) \in K[C_1, \dots, C_r]\big((x)\big)$, обладающее свойствами (под решениями понимаются решения, принадлежащие $K((x))$):

- если (9) является элементом множества $W$, то для любого продолжения $\tilde{L}$ оператора $L$ найдется его решение $\tilde{y}(x)$, для которого существуют $\tilde{C}_1, \dots, \tilde{C}_r \in K$ такие, что

$$\tilde{y}(x) = y^{\langle m \rangle}\big(x, \tilde{C}_1, \dots, \tilde{C}_r\big) + O(x^{m+1}),$$

$$\operatorname{val}\tilde{y}(x) = \operatorname{val} y(x, C_1, \dots, C_r);$$

- если $\tilde{y}(x)$ – решение некоторого продолжения $\tilde{L}$ оператора $L$, и существует элемент (9) множества $W$ такой, что выполняется

$$\operatorname{val}\tilde{y}(x) = \operatorname{val} y(x, C_1, \dots, C_r), \quad (10)$$

- то найдутся $\tilde{C}_1, \dots, \tilde{C}_r \in K$ такие, что

$$\tilde{y}(x) = y^{\langle m \rangle}\left(x, \tilde{C}_1, \dots, \tilde{C}_r\right) + O(x^{m+1});$$

- значения $m$ являются наибольшими из возможных значений, указанным образом связанных с каждым элементом множества $W$.

В множество W включаются все выражения вида (9), обладающие этими свойствами.

## 3.2 Регулярные решения

Пусть дифференциальное уравнение $\mathcal{L}(y) = 0$ имеет ненулевые решения вида (7). Алгоритмы построения таких решений обсуждаются в [5]-[13]. С помощью этих алгоритмов можно построить усечение общего регулярного решения любой заданной степени усечения $m$. Т.е. для всех рядов, входящих в решение, вычисляются коэффициенты до степени $m$, вычисленные коэффициенты могут содержать произвольные постоянные.

Для уравнения (5) с усеченными коэффициентами алгоритм, предложенный в [2], строит регулярные решения с максимально большими усечениями, водящих в них рядов, такими, что решения являются инвариантными относительно различных возможных продолжений уравнения. Входом алгоритма снова являются оператор $L \in K[x][\theta]$ и неотрицательные целые $t_0, t_1, \dots, t_r$. В результате применения алгоритма становится известным полный набор (8) степенных множителей регулярных решений, одинаковый для всех возможных продолжений оператора $L$. Для каждого допустимого множителя $x^\lambda$ строится $W(\lambda)$ – конечное множество выражений вида

$$x^\lambda \sum_{s=0}^{k} \frac{ln^s x}{s!}\left(g_{k-s}^{\langle m_s \rangle}\left((x, C_1, \dots, C_r) + O(x^{m_s+1})\right)\right), \quad (11)$$

где $g_s(x, C_1, \dots, C_r) \in K[C_1, \dots, C_r]\big((x)\big)$ для $s = 0,1, \dots, k$, обладающее свойствами:

- если (11) является элементом множества $W(\lambda)$, то для любого продолжения $\tilde{L}$ оператора $L$ найдется его решение $\tilde{y}(x) = x^\lambda \sum_{s=0}^{k} \tilde{g}_{k-s}(x)\frac{ln^s x}{s!}$, для которого существуют $\tilde{C}_1, \dots, \tilde{C}_r \in K$ такие, что

$$\tilde{g}_s(x) = g_s^{\langle m_s \rangle}\left(x, \tilde{C}_1, \dots, \tilde{C}_r\right) + O(x^{m_s+1}),$$
$$\operatorname{val}\tilde{g}_s(x) = \operatorname{val} g_s(x, C_1, \dots, C_r)$$

  для $s = 0,1, \dots, k$;

- если $\tilde{y}(x) = x^\lambda \sum_{s=0}^{k} \tilde{g}_{k-s}(x)\frac{ln^s x}{s!}$ – решение некоторого продолжения $\tilde{L}$ оператора $L$, и существует элемент (11) множества $W(\lambda)$ такой, что выполняется

$$\operatorname{val}\tilde{g}_s(x) = \operatorname{val} g_s(x, C_1, \dots, C_r), \quad (12)$$

  для $s = 0,1, \dots, k$, то найдутся $\tilde{C}_1, \dots, \tilde{C}_r \in K$ такие, что

$$\tilde{g}_s(x) = g_s^{\langle m_s \rangle}\left(x, \tilde{C}_1, \dots, \tilde{C}_r\right) + O(x^{m_s+1})$$

  для $s = 0,1, \dots, k$;

- для каждого элемента множества $W(\lambda)$ значения $m_0, m_1, \dots, m_k$ являются наибольшими из возможных, указанным образом связанных с $L$.

В множество $W(\lambda)$ включаются все выражения вида (11), обладающие этими свойствами, т.е. $W(\lambda)$ содержит полный список формул вида (11), инвариантных относительно продолжений оператора $L$.

## *4. Литералы*

Пусть задан оператор с полиномиальными коэффициентами $\boldsymbol{L}$ вида (3), набор чисел $t_0, t_1, \dots, t_r$ и пусть коэффициенты $\boldsymbol{L}$ имеют вид

$$a_i(x) = \sum_{j=0}^{t_i} a_{ij}\, x^j, i = 0,1, \dots, r$$

(если $t_i > d_i = \deg a_i(x)$, то $a_{ij} = 0$ для $j = d_i + 1, d_i + 2, \dots, t_i$). Будем говорить, что коэффициенты $a_{ij}$ *незаданы*, если $j > t_i$, для $i = 0,1, \dots, r$.

Помимо построения усечения решений (лорановых и регулярных), инвариантных относительно всех продолжений уравнения $L(y) = 0$, алгоритм позволяет прояснить влияние незаданных коэффициентов на последующие члены рядов, входящих в решения. Для незаданных коэффициентов алгоритм использует символьные обозначения, будем называть их *литералами*.

При рассмотрении продолжения $\tilde{L}$ оператора $L$ может оказаться, что $\tilde{L}$ имеет такое лораново решение $\tilde{y}(x)$, что $W$ не содержит выражения (9), для которого выполняется равенство валюаций (10). Алгоритм может определить, какие условия на незаданные коэффициенты должны быть выполнены, чтобы такое выражение появилось.

При рассмотрении продолжения $\tilde{L}$ оператора $L$ может оказаться, что $\tilde{L}$ имеет такое решение $\tilde{y}(x) = x^{\lambda} \sum_{s=0}^{k} \tilde{g}_{k-s}(x) \frac{ln^s x}{s!}$, что $W(\lambda)$ не содержит выражения (11), для которого выполняется равенство валюаций (12). Алгоритм может определить, какие условия на незаданные коэффициенты должны быть выполнены, чтобы такое выражение появилось.

Для усеченного уравнения при условии, что свободные члены всех коэффициентов известны и хотя бы один не равен нулю, полный набор степенных множителей одинаков для всех продолжений данного уравнения.

Но максимальные значения $k$ в (7) могут быть различными для разных продолжений. Алгоритм может определить, какие условия на незаданные коэффициенты должны быть выполнены, чтобы максимальные значения $k$ стали инвариантными относительно возможных продолжений уравнения.

## *5. Процедуры построения решений*

Алгоритмы построения рассматриваемых решений реализованы в системе компьютерной алгебры `Maple` ([14]) в виде процедур пакета `TruncatedSeries`[1]. Пакет предоставляет две основных процедуры:

`LaurentSolution` – построение лорановых решений;

`RegularSolution` – построение регулярных решений.

Предварительные реализации этих процедур были представлены в работах [1, 2]. За основу реализации частично взята реализация алгоритмов из пакета EG [13].

### 5.1 Аргументы и результаты работы процедур

Обе процедуры имеют одинаковые аргументы. Основные аргументы:

[1] Пакет и сессия `Maple` с примерами использования описываемых процедур доступны по адресу http://www.ccas.ru/ca/TruncatedSeries.

- Первый аргумент – дифференциальное уравнение вида (5), где $t_i \geq \deg a_i(x)$, $i = 0,1,\ldots,r$. Применение $\theta^k$ к неизвестной функции $y(x)$ записывается как `theta(y(x),x,k)`. Вместо оператора $\theta$ можно использовать и обычное дифференцирование (оператор $D = \frac{d}{dx}$); применение оператора $D^k$ к неизвестной функции $y(x)$ задается в стандартном для Maple виде `diff(y(x),x$k)`. Усеченные коэффициенты уравнения задаются в виде выражений $a_i(x) + O(x^{t_i+1})$, где $a_i(x)$ – полином степени не выше $t_i$ над полем алгебраических чисел, то есть аналогично математической записи. Нерациональные алгебраические числа в Maple представляются в виде выражения `RootOf(p(_Z), index = k)`, где `p(_Z)` – неприводимый полином, $k$-м корнем которого и является данное алгебраическое число. Например, `RootOf(Z^2-2, index=2)` $= -\sqrt{2}$.
- Второй аргумент – неизвестная функция, например, `y(x)`.

Результат работы процедуры `LaurentSolution` – список усеченных лорановых решений, из множества $W$, описанного в подразд. 3.1. Каждый элемент списка имеет вид

$$c_{v_j}x^{v_j} + c_{v_j+1}x^{v_j+1} + \cdots + c_{m_j}x^{m_j} + O\left(x^{m_j+1}\right), \tag{13}$$

где $v_j$ – валюация, для которой гарантировано существование лоранова решения при любом продолжении заданного уравнения; $m_j$ имеет прежний смысл (см. подразд. 3.1), $c_n$ – вычисленные коэффициенты лоранова решения, которые являются линейными комбинациями произвольных постоянных вида $\_c_0, \_c_1, \ldots$

Результат работы процедуры `RegularSolution` – список усеченных регулярных решений, инвариантных относительно продолжений коэффициентов заданного уравнения. Усечения содержат произвольные постоянные вида $\_c_0, \_c_1, \ldots$

Могут быть также указаны опциональные параметры:

- `'output'='literal'` – обеспечивает получения ответа не в виде списка инвариантных усечений, а в виде одного усечения с литералами. Все усеченные ряды в ответе имеют вид:

  $$c_v x^v + c_{v+1}x^{v+1} + \cdots + c_m x^m + O(x^{m+1}), \tag{14}$$

  где $v = \min v_j$, $m = 1 + \max m_j$ коэффициенты $c_n$ будут содержать литералы. Литералы представляются в виде $U_{[i,j]}$ – такой литерал соответствует незаданному коэффициенту при $x^j$ в коэффициенте исходного уравнения при $\theta^i$.
- `'degree'=n`, где n – целое число, – обеспечивает получение усечений решений заданной степени. В этом случае коэффициенты усечений возможно будут выражены через литералы. Степени построенных усечений могут быть больше заданного $n$, – должно быть вычислено по крайней мере столько коэффициентов, сколько требуется для определения всех возможных валюаций лорановых рядов, входящих в решение.

## 5.2 Примеры построения лорановых решений

1. Каждое из уравнений

$$\sin x\, \theta y(x) - x\cos x\; y(x) = 0 \tag{15}$$

$$(e^x - 1)\theta y(x) - xe^x y(x) = 0 \tag{16}$$

можно представить в виде

$$\left(x + O(x^2)\right)\theta y(x) + \left(-x + O(x^2)\right)y(x) = 0. \tag{17}$$

Применим процедуру к (17).

```
> eq1 := (x+O(x^2))*(theta(y(x),x,1))+(-x+O(x^2))*y(x);
```

$$eq1 := \left(x + O(x^2)\right)\theta(y(x), x, 1) + \left(-x + O(x^2)\right)y(x)$$

```
> LaurentSolution(eq1, y(x));
```

$$[x\_c_1 + O(x^2)]$$

Итак, имеется только одно инвариантное усечение решения с валюацией $v = 1$ и степенью усечения $m = 1$.

2. Применим процедуру к (17) еще раз, задав желаемую степень усечения, равную 2, с помощью опции `'degree'=2`:

```
> LaurentSolution(eq1, y(x), 'degree'=2);
```

$$\left[x\_c_1 + x^2\left(-\_c_1 U_{[0,2]} - \_c_1 U_{[1,2]}\right) + O(x^3)\right]$$

Итак, коэффициент решения при степени 2 зависит от литералов, то есть различные продолжения уравнения $eq1$ могут иметь разные коэффициенты решения при этой степени и найденное ранее инвариантное решение является максимально возможным.

3. Добавим к коэффициентам уравнения $eq1$ некоторые члены, соответствующие коэффициентам (15). Получим усечение решения до степени $x^2$, которое соответствует разложению в степенной ряд функции $\sin x$, являющейся решением (15):

```
> eq2 := (x+O(x^3))*(theta(y(x),x,1))+(-x+x^3/2+O(x^4))*y(x);
```

$$eq2 := \left(x + O(x^3)\right)\theta(y(x), x, 1) + \left(-x + \frac{x^3}{2} + O(x^4)\right)y(x)$$

```
> LaurentSolution(eq2, y(x));
```

$$[x\_c_1 + O(x^3)]$$

Снова $v = 1$, но $m = 2$. Легко проверить, что найденное усечение решения является продолжением инвариантного усечения решения уравнения $eq1$. При этом видно, что если подставить значения $U_{[0,2]} = 0$, $U_{[1,2]} = 0$, которые соответствуют добавленным коэффициентам, в найденное выше усечение решения уравнения $eq1$ до степени $m = 2$, то будет получено усечение решения уравнения $eq2$.

4. Теперь добавим к коэффициентам уравнения $eq1$ несколько членов, соответствующих коэффициентам (16). Получим усечение решения до степени $x^2$, которое соответствует разложению в степенной ряд функции $e^x - 1$, являющейся решением (16).

```
> eq3 := (x+x^2/2+O(x^3))*theta(y(x),x,1)+
         (-x-x^2-x^3/2+O(x^4))*y(x);
```

$$eq3 := \left(x + \frac{x^2}{2} + O(x^3)\right)\theta(y(x), x, 1) + \left(-x - x^2 - \frac{x^3}{2} + O(x^4)\right)y(x)$$

```
> LaurentSolution(eq3, y(x));
```

$$\left[x\_c_1 + \frac{x^2\_c_1}{2} + O(x^3)\right]$$

Итак, $v = 1$, $m = 2$. Вновь легко проверить, что найденное усечение решения является продолжением усечения решения уравнения $eq1$. При этом видно, что если подставить значения $U_{[0,2]} = -1$, $U_{[1,2]} = \frac{1}{2}$, которые соответствуют добавленным коэффициентам, в усечение решения уравнения $eq1$ до степени 2, то будет получено усечение решения уравнения $eq3$.

Таким образом, различные продолжения $eq2$ и $eq3$ уравнения $eq1$ дали различные инвариантные усечения решений. При этом, как и ожидалось, не возникло новых решений, инвариантные усечения решений $eq2$ и $eq3$ являются продолжениями инвариантного усечения решения $eq1$ и соответствуют его продолжению до степени 2, зависящего от литералов, с подстановкой вместо литералов соответствующих коэффициентов из уравнений $eq2$ и $eq3$.

5. Для каждого из уравнений $eq1, eq2$ и $eq3$ существует только одно значение валюации, для которого лорановы решения существуют при любом продолжении уравнения. Рассмотрим применение процедуры к следующему уравнению:

```
> eq4 := (-1+x+O(x^2))*theta(y(x),x,2)+
         (-2+O(x^2))*theta(y(x),x,1)+
         (x+O(x^2))*y(x);
```

$$eq4 := \left(-1 + x + O(x^2)\right)\theta(y(x), x, 2) + \left(-2 + O(x^2)\right)\theta(y(x), x, 1) + \left(x + O(x^2)\right)y(x)$$

```
> LaurentSolution(eq4, y(x));
```

$$\left[\_c_1 + \frac{x\_c_1}{3} + O(x^2)\right]$$

Полученный ответ означает, что существует только одно инвариантное усечение решения, оно имеет валюацию $v = 0$ и степень усечения $m = 1$.

6. Применим процедуру к $eq4$ еще раз, задав степень усечения решения, равную 3, с помощью опции `'degree'=3`:

```
> LaurentSolution(eq4, y(x), 'degree'=3);
```

$$\left[\_c_1 + \frac{x\_c_1}{3} + x^2\left(\frac{1}{12}\_c_1 + \frac{1}{8}\_c_1 U_{[0,2]}\right) + x^3\left(\frac{1}{45}\_c_1 U_{[2,2]} + \frac{1}{36}\_c_1 + \frac{23}{360}\_c_1 U_{[0,2]} + \frac{1}{45}\_c_1 U_{[1,2]} + \frac{1}{15}\_c_1 U_{[0,3]}\right) + O(x^4)\right]$$

7. Добавим к коэффициентам уравнения $eq4$ несколько коэффициентов. Применим процедуру.

```
> eq5 := (-1+x+x^2+O(x^3))*theta(y(x),x,2)+
         (-2+O(x^3))*theta(y(x),x,1)+
         (x+6*x^2+O(x^4))*y(x);
```

$$eq5 := \left(-1 + x + x^2 + O(x^3)\right)\theta(y(x), x, 2) + \left(-2 + O(x^3)\right)\theta(y(x), x, 1) + \left(x + 6x^2 + O(x^4)\right)y(x)$$

```
> LaurentSolution(eq5, y(x));
```

$$\left[\frac{\_c_1}{x^2} - \frac{5\_c_1}{x} + \_c_2 + O(x), \quad \_c_1 + \frac{x\_c_1}{3} + \frac{5x^2\_c_1}{6} + \frac{13x^3\_c_1}{30} + O(x^4)\right]$$

Полученный ответ означает, что существуют два инвариантных усечения решений: с валюацией $v_1 = 0$ и степенью усечения $m_1 = 3$, являющееся продолжением ранее найденного усечения решения для уравнения $eq4$, и второе с валюацией $v_2 = -2$ и степенью усечения $m_2 = 0$, которое является новым.

8. Добавим к коэффициентам уравнения $eq4$ несколько коэффициентов иначе. Применим процедуру.

```
>eq6 := (-1+x+x^2+O(x^3))*theta(y(x),x,2)+
        (-2+x^2+O(x^3))*theta(y(x),x,1)+
        (x+6*x^2+O(x^4))*y(x);
```

$$eq6 := \left(-1 + x + x^2 + O(x^3)\right)\theta(y(x), x, 2) + \left(-2 + x^2 + O(x^3)\right)\theta(y(x), x, 1) + \left(x + 6x^2 + O(x^4)\right)y(x)$$

```
> LaurentSolution(eq6, y(x));
```

$$\left[\_c_1 + \frac{x\_c_1}{3} + \frac{5x^2\_c_1}{6} + \frac{41x^3\_c_1}{90} + O(x^4)\right]$$

Полученный ответ означает, что вновь существует только одно инвариантное усечение решения с валюацией $v = 0$ и степенью усечения $m = 3$, являющееся продолжением ранее найденного усечения решения для $eq4$.

Видно, что различные продолжения $eq5$ и $eq6$ уравнения $eq4$ дали различные инвариантные усечения. При этом у уравнения $eq5$ возникло новое решение с другой валюацией, при этом второе инвариантное усечение решения $eq5$ и инвариантное усечение решения $eq6$ являются продолжениями инвариантного усечения решения $eq4$ и соответствуют его продолжению до степени 3 в литералах с подстановкой вместо литералов соответствующих коэффициентов из уравнений $eq5$ и $eq6$.

9. Проверим, имеет ли смысл рассматривать случай различных $t_0, t_1, \dots, t_r$, входящих в (5), или же достаточно ограничиться случаем равенства этих чисел. Иными словами, проверим может ли замена в (5) каждого $t_i$ на $t = \min_{i=0}^{r} t_i$ привести к снижению точности результата работы алгоритма.

Для следующего уравнения получаем пять начальных членов решения:

```
> eq7 := (1+O(x))*(theta(y(x),x,1))+(x^4+O(x^5))*y(x);
```

$$eq7 := \big(1 + O(x)\big)\theta(y(x), x, 1) + \big(x^4 + O(x^5)\big)y(x)$$

```
> LaurentSolution(eq7, y(x));
```

$$\left[\_c_1 - \frac{\_c_1 x^4}{4} + O(x^5)\right]$$

Если же взять $t_0 = t_1 = 0$, то получим только один начальный член решения:

```
eq8 := (1+O(x))*(theta(y(x),x,1))+ O(x)*y(x);
```

$$eq8 := \big(1 + O(x)\big)\theta(y(x), x, 1) + O(x)y(x)$$

```
LaurentSolution(eq8, y(x));
```

$$[\_c_1 + O(x)]$$

Это показывает, что при замене каждого $t_i$ на $t = \min_{i=0}^{r} t_i$ произошло снижение точности работы алгоритма. Тем самым, связанные с отказом от априорного предположения о равенстве всех $t_i$ затраты времени могут быть не напрасными.

10. Существуют уравнения, которые не имеют нетривиальных лорановых решений ни при каких продолжениях:

```
> eq9 := (2+O(x))*(theta(y(x),x,1))+(1+O(x))*y(x);
```

$$eq9 := \big(2 + O(x)\big)\theta(y(x), x, 1) + \big(1 + O(x)\big)y(x)$$

```
> LaurentSolution(eq9, y(x));
```

$$[\,]$$

Ответ – пустой список – означает отсутствие решений для всех продолжений уравнения $eq9$.

11. Процедуру можно применять к уравнениям, заданным через оператор дифференцирования $\frac{d}{dx}$.

```
>eq10 := (-x+x^2+x^3+O(x^4))*(diff(y(x),x,x))+
         (-3+x+O(x^2))*(diff(y(x),x))+
         O(x^3)*y(x);
```

$$eq10 := \big(-x + x^2 + x^3 + O(x^4)\big)\left(\frac{d^2}{dx^2}y(x)\right) + \big(-3 + x + O(x^2)\big)\left(\frac{d}{dx}y(x)\right) + O(x^3)y(x)$$

```
> LaurentSolution(eq10, y(x));
```

$$[\_c_1 + O(x^4)]$$

В случае, если не выполнено условие (4), инвариантных усечений лорановых решений не существует. В этом случае процедура возвращает `FAIL`. Следующее уравнение задано через $\frac{d}{dx}$, процедура строит эквивалентное ему уравнение через $\theta$ и определяет, что условие (4) не выполнено, следовательно, инвариантных усеченных решений не существует:

```
> eq11 := (x^2+O(x^3))*diff(y(x),x,x)+
          O(x)*diff(y(x),x)+(1+O(x))*y(x);
```

$$eq11 := \left(x^2 + O(x^3)\right)\left(\frac{d^2}{dx^2}y(x)\right) + O(x)\left(\frac{d}{dx}y(x)\right) + \left(1 + O(x)\right)y(x)$$

```
> LaurentSolution(eq11, y(x));
```

$$FAIL$$

## 5.3 Примеры построения регулярных решений

1. Применим процедуру поиска регулярных решений:

```
> eq12 := (-1+x+x^2+O(x^3))*theta(y(x),x,2)+
          (-2+O(x^2))*theta(y(x),x,1)+(O(x^4))*y(x);
```

$$eq12 := \left(-1 + x + x^2 + O(x^3)\right)\theta(y(x), x, 2) + \left(-2 + O(x^2)\right)y(x) + O(x^4)y(x)$$

```
> RegularSolution(eq12, y(x));
```

$$[\_c_1 + O(x^4)]$$

2. Добавим один дополнительный коэффициент $U_{[1,2]} = 1$ к уравнению $eq12$ и применим процедуру еще раз:

```
> eq13 := (-1+x+x^2+O(x^3))*theta(y(x),x,2)+
          (-2+x^2+O(x^3))*theta(y(x),x,1)+O(x^4)*y(x);
```

$$eq13 := \left(-1 + x + x^2 + O(x^3)\right)\theta(y(x), x, 2) + \left(-2 + x^2 + O(x^3)\right)\theta(y(x), x, 1) + O(x^4)y(x)$$

```
> RegularSolution(eq13, y(x));
```

$$\left[-\frac{\_c_1}{x^2} + \frac{4\_c_1}{x} + \_c_2 + O(x) + \ln(x)\left(\_c_1 + O(x^4)\right), \quad \_c_2 + O(x^4)\right]$$

Видим, что в этом случае возникает второе усечение регулярного решения, содержащее логарифм.

3. Применим процедуру к этому же уравнению с опцией представления результата в литералах:

```
> RegularSolution(eq13, y(x), 'output'='literal');
```

$$\begin{aligned}\ln(x)\left(\_c_1 + \frac{1}{24}x^4\_c_1 U_{[0,4]} + O(x^5)\right) &- \frac{\_c_1}{x^2} + \frac{4\_c_1}{x} + \_c_2 + x\left(\frac{2}{3}\_c_1 U_{[1,3]} - \frac{4}{3}\_c_1 U_{[2,3]}\right) \\ &+ x^2\left(\frac{1}{4}\_c_1 U_{[1,4]} - \frac{5}{12}\_c_1 U_{[1,3]} + \frac{1}{3}\_c_1 U_{[2,3]} - \frac{1}{2}\_c_1 U_{[2,4]} - \frac{1}{8}\_c_1{}_1 U_{[0,4]} + \frac{1}{8}\_c_1\right) \\ &+ x^3\left(\frac{2}{15}\_c_1 U_{[2,4]} - \frac{1}{5}\_c_1 U_{[1,4]} + \frac{2}{45}\_c_1 U_{[1,3]} - \frac{4}{45}\_c_1 U_{[2,3]} + \frac{2}{15}\_c_1 U_{[1,5]} + \frac{7}{30}\_c_1 U_{[0,4]} + \frac{1}{30}\_c_1 - \frac{4}{15}\_c_1 U_{[2,5]} - \frac{1}{15}\_c_1 U_{[0,5]}\right)\end{aligned}$$

$$+x^4\left(\frac{1}{24}\_c_1U_{[0,4]}-\frac{3}{40}\_c_1U_{[2,4]}+\frac{7}{240}\_c_1U_{[1,4]}-\frac{7}{80}\_c_1U_{[1,3]}+\frac{1}{20}\_c_1U_{[2.3]}-\frac{7}{60}\_c_1U_{[1.5]}\right.$$
$$+\frac{7}{180}\_c_1U_{[0,4]}+\frac{7}{160}\_c_1+\frac{1}{15}\_c_1U_{[2,5]}+\frac{17}{120}\_c_1U_{[0,5]}+\frac{1}{36}\_c_1U_{[1,3]}^2$$
$$-\frac{1}{36}U_{[1,3]}\_c_1U_{[2,3]}-\frac{1}{18}\_c_1U_{[2,3]}^2-\frac{1}{24}\_c_1U_{[0,6]}+\frac{1}{12}\_c_1U_{[1,6]}$$
$$\left.-\frac{1}{6}\_c_1U_{[2,6]}\right)+O(x^5)$$

4. Применим процедуру к этому же уравнению одновременно с опцией задания степени усечения:

```
> RegularSolution(eq13, y(x), 'degree'=2);
```

$$\left[-\frac{\_c_1}{x^2}+\frac{4\_c_1}{x}+\_c_2+x\left(\frac{2}{3}\_c_1U_{[1,3]}-\frac{4}{3}\_c_1U_{[2,3]}\right)\right.$$
$$+x^2\left(\frac{1}{4}\_c_1U_{[1,4]}-\frac{5}{12}\_c_1U_{[1,3]}+\frac{1}{3}\_c_1U_{[2,3]}-\frac{1}{8}\_c_1U_{[0,4]}-\frac{1}{2}\_c_1U_{[2,4]}\right.$$
$$\left.\left.+\frac{1}{8}\_c_1\right)+O(x^3)+\ln(x)\left(\_c_1+O(x^3)\right),\qquad\left(\_c_2+O(x^3)\right)\right]$$

Ответ показывает, что для получения 2-усечения как продолжения инвариантного усечения необходимо задать $U_{[0,4]}$, $U_{[1,3]}$, $U_{[1,4]}$, $U_{[2,3]}$, $U_{[2,4]}$, т.е. коэффициенты уравнения при $x^4$, $x^3\theta$, $x^4\theta$, $x^3\theta^2$, $x^4\theta^2$ соответственно.

5. Применим процедуру к этому же уравнению одновременно с опцией представления результата в литералах и опцией задания степени усечения, опции могут использоваться совместно:

```
> RegularSolution(eq13, y(x),'output'='literal', 'degree'=2);
```

$$\ln(x)\left(\_c_1+O(x^3)\right)-\frac{\_c_1}{x^2}+\frac{4\_c_1}{x}+\_c_2+x\left(\frac{2}{3}\_c_1U_{[1,3]}-\frac{4}{3}\_c_1U_{[2,3]}\right)$$
$$+x^2\left(\frac{1}{4}\_c_1U_{[1,4]}-\frac{5}{12}\_c_1U_{[1,3]}+\frac{1}{3}\_c_1U_{[2,3]}-\frac{1}{2}\_c_1U_{[2,4]}-\frac{1}{8}\_c_1U_{[0,4]}\right.$$
$$\left.+\frac{1}{8}\_c_1\right)+O(x^3)$$

6. Добавим к уравнению $eq12$ один коэффициент иначе: $U_{[1,2]}=0$.

```
> eq14 := (-1+x+x^2+O(x^3))*theta(y(x),x,2)+
          (-2+O(x^3))*theta(y(x),x,1)+
          O(x^4)*y(x);
```

$$eq14 := \left(-1+x+x^2+O(x^3)\right)\theta(y(x),x,2)+\left(-2+O(x^3)\right)\theta(y(x),x,1)$$
$$+O(x^4)y(x)$$

```
> RegularSolution(eq14, y(x));
```

$$\left[\frac{\_c_1}{x^2}-\frac{4\_c_1}{x}+\_c_2+O(x),\qquad\_c_2+O(x^4)\right]$$

Видим, что в этом случае возникает новое инвариантное регулярное решение – лораново решение с валюацией $v_2=-2$.

7. Применим процедуру к следующему уравнению

```
> eq15 := (1+x^2+O(x^3))*theta(y(x),x,3)+
          (4-x+(1/2)*x^2+O(x^3))*theta((y(x),x,2))+
          4-2*x+x^2+O(x^3))*theta(y(x),x,1+
          O(x^3)*y(x);
```

$$eq15 := \left(1 + x^2 + O(x^3)\right)\theta(y(x), x, 3) + \left(4 - x + \frac{1}{2}x^2 + O(x^3)\right)\theta(y(x), x, 2) + \left(4 - 2x + x^2 + O(x^3)\right)\theta(y(x), x, 1) + O(x^3)y(x)$$

```
> RegularSolution(eq15, y(x));
```

$$\left[\frac{\frac{21\_c_1}{16} + \frac{\_c_2}{2}}{x^2} + \frac{\_c_1}{x} + \_c_3 + O(x) + \ln(x)\left(\frac{1}{2}\frac{\_c_1}{x^2} + \_c_2 + O(x)\right) + \ln(x)^2\left(\frac{1}{2}\_c_1 + O(x^3)\right),\ \frac{1}{2}\frac{\_c_2}{x^2} + \_c_3 + O(x) + \ln(x)\left(\_c_2 + O(x^3)\right),\ \_c_3 + O(x^3)\right]$$

В данном случае существует три различных инвариантных усечения регулярных решений, содержащиеся в них ряды усечены до разных степеней, логарифм входит до степени $k = 2$.

8. Уравнение

```
> eq16 := (-1+x+O(x^3))*theta(y(x),x,2)+
          (-1-x-(3/2)*x^2+O(x^3))*theta(y(x),x,1)+
          (3/4+(1/4)*x+ (3/4)*x^2+O(x^3))*y(x);
```

$$eq16 := \left(-1 + x + O(x^3)\right)\theta(y(x), x, 2) + \left(-1 - x + \frac{3}{2}x^2 + O(x^3)\right)\theta(y(x), x, 1) + \left(\frac{3}{4} + \frac{1}{4}x - \frac{3}{4}x^2 + O(x^3)\right)y(x)$$

```
> RegularSolution(eq16, y(x));
```

$$\left[\sqrt{x}\left(-\frac{2\_c_1}{x^2} + \frac{8\_c_1}{x} + \_c_2 + O(x) + \ln(x)\left(\_c_1 + O(x^3)\right)\right),\quad \sqrt{x}\left(\_c_2 + O(x^3)\right)\right]$$

В данном случае получено регулярное решение с нецелым $\lambda$ в множителе $x^\lambda$.

9. Еще одно уравнение:

```
> eq17 := (1+O(x^2))*theta(y(x),x,3)+
          (1+2*x+O(x^2))*theta(y(x),x,2)+
          (2+x+O(x^2))*theta(y(x),x,1)+
          (2-x+O(x^2))*y(x);
```

$$eq17 := \left(1 + O(x^2)\right)\theta(y(x), x, 3) + \left(1 + 2x + O(x^2)\right)\theta(y(x), x, 2) + \left(2 + x + O(x^2)\right)\theta(y(x), x, 1) + \left(2 - x + O(x^2)\right)y(x)$$

```
> RegularSolution(eq17, y(x));
```

$$\left[\frac{\_c_1}{x} + O(x) + x^{\mathrm{RootOf}(\_Z^2+2, index=1)}\left(\_c_2 - \frac{1}{54}x\left(20 + 23\,\mathrm{RootOf}(\_Z^2 + 2, index = 1)\right)\_c_2 + O(x^2)\right) + x^{\mathrm{RootOf}(\_Z^2+2, index=2)}\left(\_c_3 - \frac{1}{54}x\left(20 + 23\,\mathrm{RootOf}(\_Z^2 + 2, index = 2)\right)\_c_3 + O(x^2)\right)\right]$$

В данном случае все продолжения уравнения имеют три неэквивалентных степенных множителя, с показателями $-1$, $\sqrt{-2}, -\sqrt{-2}$ , где $\sqrt{-2}, -\sqrt{-2}$ представлены конструкциями $\mathrm{RootOf}(\_Z^2 + 2, index = 1)$ и $\mathrm{RootOf}(\_Z^2 + 2, index = 2)$.

10. Уравнение, заданное через оператор дифференцирования $\frac{d}{dx}$:

```
> eq18 := (-x+x^2+x^3+O(x^4))*(diff(y(x),x,x))+
          (-3+x+2*x^2+O(x^3))*(diff(y(x),x))+
          O(x^3)*y(x)
```

$$eq18 := \left(-x + x^2 + x^3 + O(x^4)\right)\left(\frac{d^2}{dx^2}y(x)\right) + \left(-3 + x + 2x^2 + O(x^3)\right)\left(\frac{d}{dx}y(x)\right) + O(x^3)y(x)$$

```
> RegularSolution(eq18, y(x));
```

$$\left[-\frac{\_c_1}{x^2} + \frac{4\_c_1}{x} + \_c_2 + O(x) + \ln(x)\left(\_c_1 + O(x^4)\right), \qquad \_c_2 + O(x^4)\right]$$

В результате перехода к уравнению, записанному с помощью $\theta$, процедура получает уравнение $eq13$. Поэтому совпадают результаты вычислений.

## Список литературы / References


[1]. Абрамов С.А., Рябенко А.А., Хмельнов Д.Е. Линейные обыкновенные дифференциальные уравнения и усеченные ряды. Журнал вычислительной математики и математической физики, том 59, № 10, 2019, стр. 66–77 / Abramov S.A., Ryabenko A.A., Khmelnov D.E. Linear ordinary differential equations and truncated series. Computational Mathematics and Mathematical Physics, vol. 59, № 10, 2019.

[2]. Абрамов С.А., Рябенко А.А., Хмельнов Д.Е. Регулярные решения линейных обыкновенных дифференциальных уравнений и усеченные ряды. Журнал вычислительной математики и математической физики, том 60, № 1, 2020 / Abramov S.A., Ryabenko A.A., Khmelnov D.E. Regular solutions of linear ordinary differential equations and truncated series. Computational Mathematics and Mathematical Physics, vol. 60, № 1, 2020.

[3]. Moulay Barkatou, Eckhard Pflügel. An algorithm computing the regular formal solutions of a system of linear differential equations. Journal of Symbolic Computation. vol. 28, issues 4–5, 1999, pp. 569-587.

[4]. Sergei A. Abramov, Manuel Bronstein, Marko Petkovšek. On polynomial solutions of linear operator equations. In Proc. of the 1995 international symposium on Symbolic and algebraic computation. 1995, pp. 290-296.

[5]. Ferdinand Georg Frobenius. Integration der linearen Differentialgleichungen mit verânder Koefficienten. Journal für die reine und angewandte Mathematik, vol.76, 1873, pp. 214-235.

[6]. Lothar Heffter. Einleitung in Die Theorie Der Linearen Differentialgleichungen Mit Einer Unabhängigen Variablen. Teubner, Leipzig, 1894, 283 p.

[7]. Évelyne Tournier. Solutions formelles d'équations différentielles. Le logiciel de calcul formel DESIR, Étude théorique et realization. Thèse d'État, Université I de Grenoble, 1987.

[8]. Eckhard Pflügel. DESIR-II. RT 154, IMAG Grenoble. 1996.

[9]. Abramov S., Bronstein M., Khmelnov D. On regular and logarithmic solutions of ordinary linear differential systems. Lecture Notes in Computer Science, vol. 3718, 2005, pp. 1-12.

[10]. Abramov S.A., Barkatou M.A., Pfluegel E. Higher-order linear differential systems with truncated coefficients. Lecture Notes in Computer Science, vol. 6885, 2011, pp. 10-24.

[11]. Abramov S.A., Barkatou M.A. Computable Infinite Power Series in the Role of Coefficients of Linear Differential Systems. Lecture Notes in Computer Science, vol. 8660, 2014, pp. 1-12.

[12]. Abramov S.A., Khmelnov D.E. Regular solutions of linear differential systems with power series coefficients. Programming and Computer Software, vol. 40, issue 2, 2014, pp. 98–10.

[13]. Abramov S.A., Ryabenko A.A., Khmelnov D.E. Procedures for searching local solutions of linear differential systems with infinite power series in the role of coefficients. Programming and Computer Software, vol. 42, issue 2, 2016, pp. 55–64.

[14]. Maple online, available at: http://www.maplesoft.com/support/help/.

## Информация об авторах / Information about authors

Сергей Александрович АБРАМОВ, доктор физико-математических наук, профессор, главный научный сотрудник. Научные интересы: компьютерная алгебра; символьные вычисления; символьное суммирование; линейные обыкновенные дифференциальные уравнения; линейные (q-)разностные уравнения; доказательство тождеств.

Sergey Alexandrovich ABRAMOV, Doctor of Physical and Mathematical Sciences, Professor, Chief Researcher. Research interests: computer algebra; symbolic calculations; symbolic summation; linear ordinary differential equations; linear (q-) difference equations; proof of identities.

Анна Андреевна РЯБЕНКО, кандидат физико-математических наук, научный сотрудник. Научные интересы: компьютерная алгебра; символьные вычисления; линейные обыкновенных дифференциальные и разностные уравнения и их системы.

Anna Andreevna RYABENKO, Candidate of Physical and Mathematical Sciences, Researcher. Research interests: computer algebra; symbolic calculations; linear differential and difference equations and systems, summation problems.

Денис Евгеньевич ХМЕЛЬНОВ, кандидат физико-математических наук, младший научный сотрудник. Научные интересы: компьютерная алгебра; символьные вычисления; особые точки систем линейных обыкновенны дифференциальных уравнений с полиномиальными коэффициентами

Denis Evgenevich KHMELNOV, Candidate of Physical and Mathematical Sciences, Junior Researcher. Research interests: computer algebra; symbolic calculations; singular points of linear differential systems with polynomial coefficients.