ТРУДЫ

ИНСТИТУТА СИСТЕМНОГО ПРОГРАММИРОВАНИЯ РАН

PROCEEDINGS OF THE INSTITUTE FOR SYSTEM PROGRAMMING OF THE RAS

ISSN Print 2079-8156
Том 32 Выпуск 2

ISSN Online 2220-6426
Volume 32 Issue 2

Институт системного
программирования
им. В.П. Иванникова РАН

Москва, 2020

ИСП РАН

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

Труды Института системного программирования РАН
Proceedings of the Institute for System Programming of the RAS

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

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

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

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

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

Труды ИСП РАН реферируются и/или индексируются в:

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

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

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

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

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

Google Scholar, ULRICHSWEB Global Serials Directory, Russian Science Citation Index, OCLC WorldCat, BASE, OpenDOAR, CYBERLENINKA, DOAJ Directory of Open Access Journals, eLIBRARY.RU, ROAR Registry of Open Access Repositories, Crossref, Math-Net.Ru

УДК004.45

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

Address: 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
Tel: +7(495) 912-44-25
E-mail: proceedings@ispras.ru
Web: https://ispranproceedings.elpub.ru/

Содержание

Table of Contents

DOI: 10.15514/ISPRAS-2020-32(2)-1

Векторные модели на основе символьных н-грамм для морфологического анализа текстов

Ц.Г. Гукасян, ORCID: 0000-0003-2389-517X <tsggukasyan@ispras.ru>
Российско-Армянский университет,
ул. Овсепа Эмина 123, Ереван, 119991 РА

Аннотация. В работе представляются модификации модели векторов fastText, основанные исключительно на н-граммах, для морфологического анализа текстов. fastText - библиотека для классификации текстов и обучения векторных представлений. Представление каждого слова вычисляется как сумма его отдельного вектора и векторов его символьных н-грамм. fastText хранит и использует отдельный вектор для целого слова, но во внесловарных случаях такой вектор отсутствует, что приводит к ухудшению качества получаемого вектора слова. Кроме того, в результате хранения векторов для целых слов, модели fastText обычно требуют много памяти для хранения и обработки Это становится особенно проблематично для морфологически богатых языков, учитывая многочисленность словоформ. В отличие от исходной модели fastText, предлагаемые варианты используют только информацию об н-граммах слова, избавляя от зависимости от векторов на уровне слов и в то же время помогая значительно сократить количество параметров в модели. Предлагается два способа извлечения информации из слова: внутренние символьные н-граммы и суффиксы. Модели тестируются на корпусе СинТагРус в задаче морфологической разметки и лемматизации русского языка, и показывают результаты, сравнимые с исходной моделью fastText.

Ключевые слова: вектора слов; морфологический анализ; lemmatization

Для цитирования: Гукасян Ц.Г. Векторные модели на основе символьных н-грамм для морфологического анализа текстов**.** Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 7-14. DOI: 10.15514/ISPRAS–2020–32(2)–1

Благодарности. Автор благодарит Ешилбашян Е.М., Аветисяна К.И., Королева С.Н. и Майорова В.Д. за помощь в разработке и экспериментах, а также Турдакова Д.Ю., Андрианова И.А., Хачатряна Г.А., Трифонова В.Д. за ценные отзывы и обсуждения.

Character N-gram-Based Word Embeddings for Morphological Analysis of Texts

Ts. Ghukasyan, ORCID: 0000-0003-2389-517X <tsggukasyan@ispras.ru>
Ivannikov Laboratory for System Programming at Russian-Armenian University,
123 Hovsep Emin str., Yerevan, 0051 Armenia

Abstract. The paper presents modifications of fastText word embedding model based solely on n-grams, for morphological analysis of texts. fastText is a library for classifying texts and teaching vector representations. The representation of each word is calculated as the sum of its individual vector and the vectors of its symbolic n-grams. fastText stores and uses a separate vector for the whole word, but in extra-vocabular cases there is no such vector, which leads to a deterioration in the quality of the resulting word vector. In addition, as a result of storing vectors for whole words, fastText models usually require a lot of memory for storage and processing. This becomes especially problematic for morphologically rich languages, given the large number of word forms. Unlike the original fastText model, the proposed modifications only pretrain and use vectors for the character n-grams of a word, eliminating the reliance on word-level vectors and at the same time

helping to significantly reduce the number of parameters in the model. Two approaches are used to extract information from a word: internal character n-grams and suffixes. Proposed models are tested in the task of morphological analysis and lemmatization of the Russian language, using SynTagRus corpus, and demonstrate results comparable to the original fastText.

Keywords: word embeddings; morphological analysis; lemmatization

For citation: Ghukasyan Ts. Character N-gram-Based Word Embeddings for Morphological Analysis of Texts. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 7-14 (in Russian). DOI: 10.15514/ISPRAS–2020–32(2)–1

Acknowledgments. The author thanks E. Yeshilbashyan, K. Avetisyan, S. N. Korolev and Mayorov V.D. for assistance in the development and experiments, as well as Turdakov D.Yu., Andrianov I.A., Khachatryan G.A., Trifonov V.D. for valuable feedback and discussions.

1. Введение

Вектора слов широко и успешно используются во многих задачах обработки естественного языка, но они имеют серьезные недостатки для обработки редких слов или слов из словарного запаса, для которых вложения либо недоступны, либо неудовлетворительны. Это особенно проблематично для морфологически богатых языков, где лексемы имеют много редких словоформ. Векторные представления слов на основе подслов были популярным направлением исследований в последние годы. В них слово рассматривается как мешок подслов, и используется исключительно эта внутренняя информация для составления вектора целого слова, позволяя получать вектора для внесловарных слов и тем самым помогая преодолеть проблему разреженности данных. Ранние попытки были сосредоточены на восстановлении предобученных векторов [1-4]. Однако, эти подходы по-прежнему требовали предварительного обучения векторов на уровне слов и были разработаны специально для обработки внесловарных случаев. Последующие подходы были направлены на обучение векторов морфемоподобных подслов непосредственно. В этих работах получение вектора слова производилось через агрегирование векторов подслов с помощью обычного усреднения, рекуррентных сетей или механизма self-attention [5-6].

fastText [7] – библиотека для классификации текстов и обучения векторных представлений. В последнем режиме fastText учит представления слов с использованием символьных н-грамм, обучая нейронную сеть вида SkipGram или CBOW на неразмеченных текстах. Представление каждого слова вычисляется как сумма его отдельного вектора и векторов его символьных н-грамм. Отсюда вытекает преимущество fastText по сравнению с другими моделями встраивания слов, заключающееся в том, что он может вычислять представление для слова вне словарного запаса (OOV), используя его символьные н-граммы. Как видно, fastText хранит и использует отдельный вектор для целого слова, но во внесловарных случаях такой вектор отсутствует, что приводит к ухудшению качества получаемого вектора слова. Кроме того, в результате хранения векторов для целых слов, модели fastText обычно требуют много памяти для хранения и обработки (модели векторов от Facebook, обученные на Common Crawl, весят 7,3 ГБ и 4,5 ГБ в форматах .bin и .vec соответственно [8]). Это становится особенно проблематично для морфологически богатых языков, учитывая многочисленность словоформ. В результате, для таких языков получаются модели, имеющие большое количество параметров и требующие много памяти. По сравнению со словоформами, словарь подслов имеет, как правило, меньший размер и позволяет получать модели со значительно более маленьким числом параметров.

В этой работе рассматриваются модификации fastText, которые удаляют вектора на уровне слов из модели и основываются только на символьных н-граммах для обучения и генерации представлений. За исключением использования только н-грамм при вычислении вектора слов, представленные модели в остальном не отличаются от fastText. Предложенные модели

тестируются в задаче морфологического анализа и лемматизации русских текстов и показываем, что они демонстрируют результаты, сопоставимые с исходной версией fastText.

2. Обзор моделей

В этом разделе описаны модель векторов fastText и две ее модификации – no-fastText и so-fastText, позволяющие получать вектора на основе символьных н-грамм.

2.1 fastText

В режиме без учителя fastText обучает представления слов. При обучении используется нейронная сеть с архитектурой CBOW или SkipGram [13]. SkipGram принимает в качестве входных данных векторное представление слова и применяет полносвязанный слой с softmax активацией для прогнозирования его контекстных слов. Сеть хранит отдельный вектор для каждого слова и ограниченное количество векторов для н-грамм символов. Представление каждого слова вычисляется как сумма его вектора и векторов н-грамм его символов:

$$\mathrm{V} = \mathrm{w}^{\mathrm{T}}\mathrm{E}_{\mathrm{w}} + \sum_{\mathrm{k=min}}^{\mathrm{max}} \sum_{\mathrm{i=1}}^{\mathrm{n-k+1}} \mathrm{s_{ki}}^{\mathrm{T}}\mathrm{E}_{\mathrm{s}} \quad (1)$$

$$\boldsymbol{Output} = \boldsymbol{V} * \boldsymbol{E'} \quad \textbf{(2)}$$

где W – множество слов; S – множество символьных н-грамм; $w \in \mathbb{Z}_{+}^{|W|\mathrm{x}1}$ – унитарный код слова; $s_{ki\in} \in \mathbb{Z}_{+}^{|S|\mathrm{x}1}$ – унитарный код символьной н-граммы; $E_w \in \mathbb{R}^{|\mathrm{W}|\mathrm{xdim}}$ и $E_S \in \mathbb{R}^{|\mathrm{S}|\mathrm{xdim}}$ – матрицы векторов слов и символьных н-грамм соответственно; k – длина н-граммы (min и max – гиперпараметры); $E' \in \mathbb{R}^{\mathrm{dimx}|\mathrm{W}|}$ – параметры выходного слоя. В реализации модели символьные н-граммы отображаются во множество меньшего размера с помощью хеширования. Когда встречается незнакомое слово, его представление вычисляется как сумма векторов н-грамм.

В fastText вместо softmax выходные значения обрабатываются отдельно, как в задаче бинарной классификации. Матрицы векторов и параметры выходного слоя обучаются путем обратного распространения ошибки с использованием негативного семплирования и стохастического градиентного спуска.

2.2. Ngrams-only fastText

В ngrams-only fastText (no-fastText), первой рассматриваемой модификации fastText, во время обучения и генерации представления слова игнорируется вектор целого слова и учитываются только н-граммы символов. Вектор вычисляется следующим образом:

$$\boldsymbol{V} = \sum_{\boldsymbol{k=min}}^{\boldsymbol{max}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n-k+1}} \boldsymbol{s_{ki}}^{\boldsymbol{T}}\boldsymbol{E_s} \quad (\mathbf{3})$$

В остальном модель идентична fastText. Данная модель основана на предположении, что символьные н-граммы слова несут достаточно информации, чтобы восстановить его значение. По сравнению со словами, словарный запас символов н-грамм ограничен, и с использованием хеширования они отображаются в фиксированное число векторов. Следовательно, модифицированная модель имеет значительно меньше параметров, чем исходная.

2.3 Suffixes-only fastText

Вторая предложенная модификация (suffixes-only fastText; so-fastText) аналогична первой, но сжимает модель еще больше. Она исключает внутренние н-граммы символов из модели, оставляя только н-граммы, которые в конце слова, то есть суффиксы. Представление слова вычисляется как сумма векторов его суффиксов s_k:

$$\boldsymbol{V} = \sum_{\boldsymbol{k=min}}^{\boldsymbol{max}} \boldsymbol{s_k}^{\boldsymbol{T}} \boldsymbol{E_s} \qquad (\mathbf{4})$$

Предположение, лежащее в основе этой модели, заключается в том, что суффиксы могут содержать достаточно информации о форме слова, необходимой для морфологических задач [14]. Несмотря на то, что изменения позволяют еще больше сократить словарный запас, с учетом реализации и использования хеширования количество параметров остается неизменным по сравнению с no-fastText. В то же время, в этой модификации теоретически менее выражена проблема коллизий из fastText и no-fastText, где разные н-граммы отображаются в один и тот же вектор.

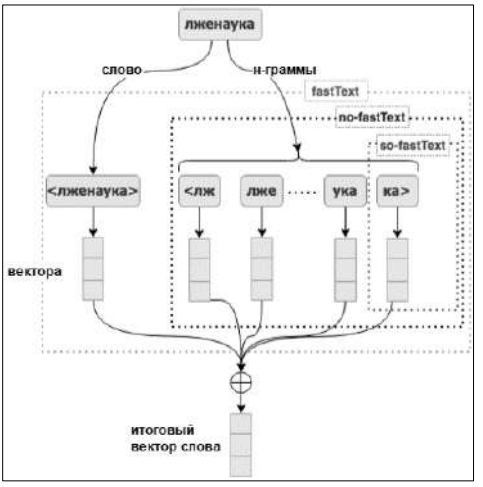

Рис. 1. Вычисление вектора слова в моделях fastText, no-fastText, so-fastText
Fig. 1. Word vector calculation in fastText, no-fastText, so-fastText models

3. Эксперименты

В этом разделе описаны параметры экспериментов по определению эффективности предложенных изменений в модели векторов fastText. Были проведены эксперименты для оценки производительности no-fastText и so-fastText в задаче морфологического анализа и лемматизации русского языка по сравнению с оригинальной версией fastText.

3.1 Векторные представления

Для всех векторов был использован размер 200 вместо значения по умолчанию 100, так как CoNLL Shared Task 2018 показала, что более высокие размеры лучше подходят для морфологического анализа [15]. В символьных н-граммов во всех моделях использовалась минимальная длина 3 и максимальная длина 6, и было задано ограничение 100000 для общего числа их векторов. Также, словарь был ограничен 400000 наиболее частыми словами. Для остальных параметров оставили значения по умолчанию.

3.2 Архитектура анализатора

Для морфологического анализатора и лемматизации была использована нейронная сеть, в целом придерживающаяся архитектуре COMBO [16] из CoNLL Shared Task 2018, с небольшими изменениями, как использование highway-слоев [17] для признаков на основе символов слова.

В качестве входных данных используются 2 компонента: вектор слова и результаты 3-слойной расширяющейся сверточной сети над символами.

Входные данные обрабатываются двумя двунаправленными рекуррентными слоями на уровне предложения, затем их результат передается в отдельные слои для конкретных задач. Для лемматизатора используется 3-слойная расширяющаяся сверточная сеть, которая выдает отдельный вектор признаков для каждого символа леммы. Лемма составляется конкатенацией отдельно предсказанных символов на основе этих векторов признаков. Прогнозирование морфологических признаков выполняется гранулярно, используя отдельный полносвязный слой для каждого из них. Важно отметить, что в этой сети параметры типичной предобученной модели векторов, как fastText или GloVe, будут составлять почти 90% от общего количества.

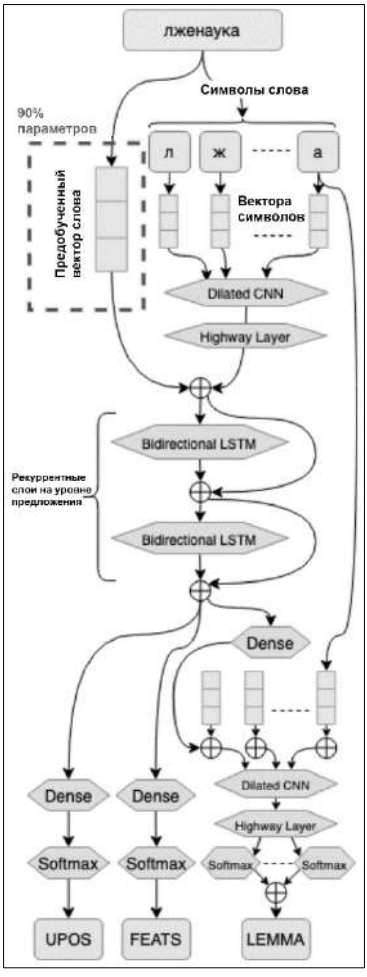

Рис. 2. Архитектура анализатора
Fig. 2. Analyzer architecture

Для реализации, обучения и оценки сети использовалась платформа BabylonDigger[1]. Во время экспериментов в сеть передавались готовые вектора, предварительно обученные и сгенерированные для каждого слова из обучающего, валидационного и тестового наборов. Эти вектора не обновлялись во время обучения, которое длилось 100 эпох, с использованием размера пакета 25 и оптимизатора Adam [18] с начальной скоростью обучения 0.01 и линейным затуханием. Каждый эксперимент был повторен с 10 различными случайными инициализациями параметров сети, и результаты были усреднены.

3.3 Данные

Для обучения векторных представлений был использован набор текстовых данных из Википедии и Common Crawl, взятый с сайта CoNLL Shared Task 2017 [15]. Для обучения анализатора использовался синтаксически размеченный корпус русского языка SynTagRus [19] версии Universal Dependencies v2.4.

4. Результаты

Табл. 1. Точность морфологического анализа (UPOS, FEATS) и лемматизации (LEMMA) с использованием разных моделей векторного представления слов
Table 1. The accuracy of morphological analysis (UPOS, FEATS) and lemmatization (LEMMA) using different models of vector representation of words

	Accuracy		
Модель	**UPOS**	**FEATS**	**LEMMA**
fastText	98.00	93.70	**96.49**
no-fastText	**98.02**	**93.72**	96.31
so-fastText	97.75	92.88	92.88

Представленные модели демонстрируют сопоставимый уровень точности по сравнению с исходной версией fastText, при этом no-fastText даже немного опережает в категориях UPOS и FEATS (Таблица 1). Несмотря на строгое ограничение на использование только суффиксов, вектора so-fastText продемонстрировали относительно хорошие результаты для русского языка [20]. В то же время нужно отметить, что эта модель исключает значительный объем информации, которая могла бы понадобиться в семантических задачах, где применяются векторные представления слов [21].

Табл. 2. Размер и количество параметров в моделях векторного представления слов
Table 2. The size and number of parameters in the models of the vector embedding of words

Модель	**.bin (Мб)**	**.txt* (Мб)**	**Количество параметров x10^6**
fastText	730	694	100
(no/so)-fastText	406	167.1	20

*текстовый файл, в каждой строке которого хранятся значения вектора слов/подслов

Благодаря небольшому размеру словаря, модели no-fastText и so-fastText заметно легче, чем обычные вложения на уровне слов, с точки зрения количества параметров и памяти (табл. 2). В них общее количество параметров меньше в 5 раз. Также, хранение параметров, необходимых для вычисления вектора слова, в текстовом формате требует более чем 4 раза меньше памяти в модифицированных моделях, чем в исходной версии.

[1] https://github.com/ispras-texterra/babylondigger

5. Заключение

В работе были представлены модификации модели векторного представления слов fastText, которые используют только н-граммы символов для генерации представления слова. Помимо внутренних н-грамм, также была рассмотрена модель, где используются только суффиксы слова. Эти модификации устраняют зависимость fastText от векторов целых слов, и приводят к более легким моделям с точки зрения числа параметров. Модели были протестированы в задаче морфологического анализа и лемматизации русского языка, где продемонстрировали уровень точности, сравнимый с исходной моделью.

Список литературы

[1]. Pinter Y., Guthrie R., Eisenstein J. Mimicking Word Embeddings using Subword RNNs. In Proc. of the 2017 Conference on Empirical Methods in Natural Language Processing, 2017, pp. 102-112.
[2]. Schick T., Schütze H. Attentive Mimicking: Better Word Embeddings by Attending to Informative Contexts. In Proc. of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, vol. 1 (Long and Short Papers), 2019, pp. 489-494.
[3]. Zhao J., Mudgal S., Liang Y. Generalizing Word Embeddings using Bag of Subwords. In Proc. of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018, pp. 601-606.
[4]. Sasaki S., Suzuki J., Inui K. Subword-based Compact Reconstruction of Word Embeddings. In Proc. of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, vol. 1 (Long and Short Papers), 2019, pp. 3498-3508.
[5]. Heinzerling B., Strube M. BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages. In Proc. of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), 2018, pp. 2989-2993.
[6]. Zhu Y., Vulić I., Korhonen A. A Systematic Study of Leveraging Subword Information for Learning Word Representations. In Proc. of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, vol. 1 (Long and Short Papers), 2019, pp. 912-932.
[7]. Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov. Enriching Word Vectors with Subword Information. Transactions of the Association for Computational Linguistics, vol. 5, 2017, pp. 135–146.
[8]. Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, Tomas Mikolov. Learning Word Vectors for 157 Languages. In Proc. of the Eleventh International Conference on Language Resources and Evaluation, 2018, pp. 3483-3487.
[9]. Shibata Y. et al. Byte Pair encoding: A text compression scheme that accelerates pattern matching. Technical Report DOI-TR-161, Department of Informatics, Kyushu University, 1999.
[10]. Pennington J., Socher R., Manning C. D. Glove: Global vectors for word representation. In Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing, 2014, pp. 1532-1543.
[11]. Üstün A., Kurfalı M., Can B. Characters or Morphemes: How to Represent Words? In Proc. of the Third Workshop on Representation Learning for NLP, 2018, pp. 144-153.
[12]. Devlin J. et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding In Proc. of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, vol. 1 (Long and Short Papers), 2019, pp. 4171-4186.
[13]. Mikolov T. et al. Advances in Pre-Training Distributed Word Representations. In Proc. of the Eleventh International Conference on Language Resources and Evaluation, 2018, pp. 52-55.
[14]. Zhu Y. et al. On the Importance of Subword Information for Morphological Tasks in Truly Low-Resource Languages. In Proc. of the 23rd Conference on Computational Natural Language Learning (CoNLL), 2019, pp. 216-226.
[15]. Zeman D. et al. CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies. In Proc. of the CoNLL 2018 Shared Task: Multilingual parsing from raw text to universal dependencies, 2018, pp. 1-19.
[16]. Rybak P., Wróblewska A. Semi-supervised neural system for tagging, parsing and lemmatization In Proc. of the CoNLL 2018 Shared Task: Multilingual parsing from raw text to universal dependencies, 2018, pp. 45-54.
[17]. Srivastava R. K., Greff K., Schmidhuber J. Highway networks. arXiv preprint arXiv:1505.00387, 2015.

[18]. Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Proc. of the 3rd International Conference on Learning Representations, 2015, pp. 1-15.

[19]. Boguslavsky I. SynTagRus – a Deeply Annotated Corpus of Russian. In Les émotions dans le discours – Emotions in Discourse, Peter Lang GmbH, Internationaler Verlag der Wissenschaften, 2014, pp. 367-380.

[20]. Турдаков Д., Астраханцев Н., Недумов Я., Сысоев А., Андрианов И., Майоров В., Федоренко Д., Коршунов А., Кузнецов С. Texterra: инфраструктура для анализа текстов. Труды ИСП РАН, том 26, вып. 1, 2014 г., стр. 421-438 / Turdakov D., Astrakhantsev N., Nedumov Y., Sysoev A., Andrianov I., Mayorov V., Fedorenko D., Korshunov A., Kuznetsov S. Texterra: A Framework for Text Analysis. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 1, 2014, pp. 421-438 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-18.

[21]. Андрианов И.А., Майоров В.Д., Турдаков Д.Ю. Современные методы аспектно-ориентированного анализа эмоциональной окраски. Труды ИСП РАН, том 27, вып. 5, 2015 г., стр. 5-22 / Andrianov I.A., Mayorov V.D., Turdakov, D.Y. Modern approaches to aspect-based sentiment analysis. Trudy ISP RAN/ Proc. ISP RAS, vol. 27, issue 5, 2015, pp. 5-22 (in Russian). DOI: 10.15514/ISPRAS-2015-27(5)-1

Информация об авторе / Information about the author

Цолак Гукасович ГУКАСЯН является аспирантом кафедры системного программирования Российско-Армянского университета. Его научные интересы включают обработку естественного языка, машинное обучение.

Tsolak Gukasovitch GHUKASYAN is a postgraduate student of the Department of System Programming of Russian-Armenian University. His research interests include natural language processing, machine learning.

DOI: 10.15514/ISPRAS-2020-32(2)-2

Assessing the quality of the requirements specification by applying GQM approach and using NLP tools

E.V. Timoshchuk, ORCID 0000-0001-9985-7036 <timoshchuk.ev@pm.me>
Peter the Great St.Petersburg Polytechnic University,
29, Polytechnicheskaya, St.Petersburg, 195251, Russia

Abstract. Software requirements are quite difficult to measure in terms of quality without reviews and subjective opinions of stakeholders. Quality assessment of specifications in an automated way saves project resources and prevents future latent defects in software. Requirements quality can be evaluated based on a huge variety of attributes, but their meaning is quite vague without any mapping to specific measurement metrics. Application of goal-question-metric (GQM) approach in the quality model helps to choose the most important quality attributes and create a mapping with metrics, which can be collected and calculated automatically. Text of software requirements written in natural language can be analyzed by NLP tools due to identify weak signle words and phrases, which make statements ambiguous. Metrics for such quality attributes as ambiguity, singularity, subjectivity, completeness, and readability are proposed in this work. The quality model was implemented in a prototype by adopting natural language processing techniques for requirements written in the Russian language with the support of external API.

Keywords: requirements quality; GQM approach; quality assessment; Natural Language Processing

For citation: Timoshchuk E.V. Assessing the quality of the requirements specification by applying GQM approach and using NLP tools. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 15-28. DOI: 10.15514/ISPRAS–2020–32(2)–2

Acknowledgments. Author wishes to thank the organizers of CASE in Tools International Hackathon and every single person who contributed to the success of this event. This research would not have been conducted without efforts of Konstantin Valeev, the challenge owner, who shed more light on grey areas of this project and provided this research with enough resources.

Оценка качества требований к программному обеспечению с применением метода GQM и инструментов обработки естественного языка

Е.В. Тимощук, ORCID 0000-0001-9985-7036 <timoshchuk.ev@pm.me>
Санкт-Петербургский политехнический университет Петра Великого,
195251, Россия, Санкт-Петербург, ул. Политехническая, д. 29

Аннотация. Требования к программному обеспечению достаточно трудно объективно измерить по критериям качества без вовлечения в процесс оценки непосредственно самих заинтересованных сторон. Оценка качества документов спецификации требований в автоматическом режиме может существенно сократить расходы бюджета проекта и также предотвращает появление скрытых дефектов в программном обеспечении на более поздних этапах разработки. Качество требований может быть оценено, основываясь на широком разнообразии атрибутов качества, но значение каждого такого атрибута достаточно расплывчато и не имеет строгой привязки к какой-либо измеримой метрике. Использование метода GQM (Goal-Question-Metric) в процессе построения модели оценки

может помочь выявить наиболее важные критерии качества и установить связь между атрибутами и конкретными метриками, которые могут быть собраны и вычислены автоматически. Текст требований к программному обеспечению, написанный на естественном языке, может быть проанализирован при помощи инструментов NLP (Natural Language Processing) с целью выявления наиболее слабых слов и фраз, которые делают предложения неоднозначными и двусмысленными. Метрики для таких критериев качества как неоднозначность, единственность, субъективность, полнота и удобочитаемость предложены в данной работе. Описанная модель оценки качества была реализована при помощи сторонних инструментов с открытым исходным кодом для требований, написанных на русском языке.

Keywords: качество требований; метод GQM; оценка качества; обработка естественного языка

Для цитирования: Тимощук Е.В. Оценка качества требований к программному обеспечению с применением метода GQM и инструментов обработки естественного языка. Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 15-28 (на английском языке). DOI: 10.15514/ISPRAS–2020–32(2)–2

Благодарности. Автор хотел бы поблагодарить организаторов CASE in Tools International Hackathon и всех, кто внес свой вклад в успех этого мероприятия. Это исследование не было бы выполнено без поддержки Константина Валеева, постановщика задачи, который помог понять суть проекта и обеспечил исследование достаточными ресурсами.

1. Introduction

Low quality of requirements may cause expensive consequences during the software development lifecycle. Issues in requirements, such as ambiguity and incompleteness may lead to time and cost overrun in the project. Especially, if iterations are long and feedback comes too late – the faster a problem is found, the cheaper it is to fix. However, it is not so easy to properly detect in automated way whether a requirements specification has lack of clarity. Some of these issues require specific domain knowledge to be uncovered. For example, it is very difficult to detect with automatic approaches whether a requirements specification is lacking necessary features.

There are a variety of requirements management techniques, tools, and practices in the software development field. However, they should be tailored to the choosen development methods. Requirements engineering assumes that the requirements must meet a number of criteria. Descriptions of such criteria can be found in both the scientific and methodological literature and put on standards. For instance, the IEEE standard [1] for requirements engineering defines quality attributes for a single requirement: necessary, appropriate, feasible, verifiable, correct, conforming, complete, consistent, comprehensible etc. Several language criteria are also defined for the text of requirements. Unbounded and ambiguous terms should be avoided. Requirements should state ‘what’ is needed, not ‘how’.

Despite the exact techniques on how to gather and validate this requirements quality metrics are not formulated by the standard (which is obviously beyond its area of consideration), but they are the topic for various researches.

Software requirements in industry are most often written in natural language which has no any formal semantics. This is the main reason why issues in requirements are so hard to detect. Approach that is presented in this paper faces the problem of fast feedback and getting some knowledge about specification’s semantics with concrete symptoms for a requirement artefact’s quality defect.

Natural language processing (NLP) tools and systems have been applied to analysing requirements texts since the 1980’s [2]. More and more NLP systems and tools with applying requirements evaluation are developed in recent years. It is a most appropriate technique to analyse human text and collect some useful data about it.

Goal-question-metric (GQM) involves defining achievable goals in order to attain quality thereby providing questions in relation to how to achieve the goals and metrics are provided to ascertain the progress in attaining set goals. This research work makes use of GQM by setting goals such as unambiguity, completeness, readability that the requirements must meet, questions on how to derive these quality attributes, and what to measure in determining if our requirements match defined goals.

The ultimate goal of this work was encapsulating the best of these techniques and methods for measurement requirement quality into a single model and provide a prototype of tool for automated validation of real-world requirements against it with Russian language support. This paper will present some measuring quality indicators for natural language requirements presented in textual natural format. Identified quality indicators first of all should point out concrete defects and provide suggestions for improvements. Proposed tool prototype combined all this indicators and computes quality measures in a fully automated way.

This paper describes a workflow (see, e.g. fig 1) of a model that helps to obtain low-level quality indicators based on some metrics of textual requirements, such as text length, number of ambiguous terms, imperative verbal forms, etc. This model has been implemented in a tool that computes quality metrics in a fully automated way.

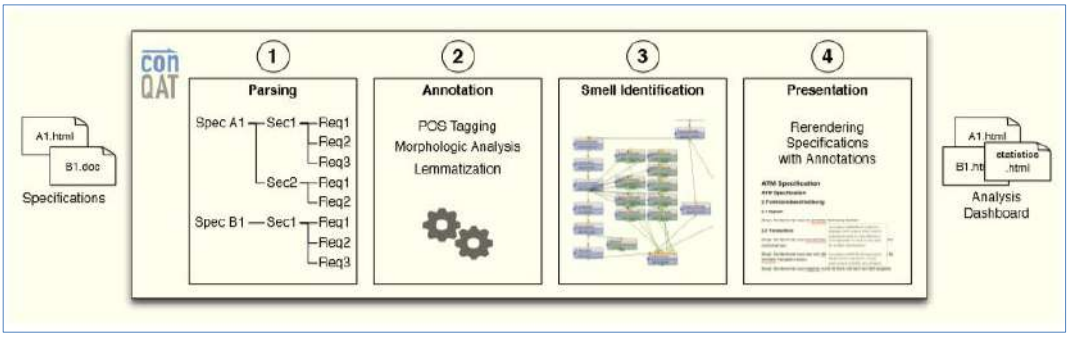

Fig. 1. Example of requirement analysis workflow for automated tool conQAT [3]

2. Related studies

Many publications discuss different problems in requirements specifications. First problem is automatization of assessment process. Mostly researchers focus on approaches for automated detection of specific defects in requirements specifications without any additional interaction with user. The main features of such tools include detection of similarity in requirements [4] and ambiguity [5], detection of missing information, linguistic flaws and passive sentences [6].

K¨orner and Brumm presented RESI, a tool that scans documents for linguistic flaws and reports them to the user (see Section II-C). It can be used to detect defects in requirements specifications, but the high number of false positives results prohibits the actual use of this tool in a real case [7].

Fabbrini et al. presented QuARS (Quality Analyzer for Requirement Specifications) tool that checks requirements specification by comparing with predefined word lists [5]. The lists give indicators for problems and if the number of indicators in the phrase exceeds a given threshold, requirements is ambigious.

Verma et al. presented their RAT (Requirements Analysis Tool) [8] tool - a word processor that able to analyze natural language requirements based on a user-defined glossary and constrained language. RAT highlights problematic requirements directly in the requirements specifications, but this process requires some training by real users.

Goldin and Berry [9] implemented a tool called Abstfinder to identify the abstractions from natural language text used for requirements elicitation.

Lee and Bryant [10] developed an automated system to assist the engineers to build a formal representation from informal requirements.

Overhelming majority of authors suppose that ambiguity carries a high risk of misunderstanding among different readers. Several studies dealing with ambiguity identification have aimed to help

improve the quality of requirements documents. Some tools have been developed specifically to detect, measure and reduce possible structural ambiguities in text.

In paper their paper, Yang H, Willis A, De Roeck A and Nuseibeh B describe an automated approach for characterizing and detecting ambiguities that was implemented in NAI (Nocuous Ambiguity Identification) tool prototype [11]. Implemented tool uses machine learning algorithm to determine whether an ambiguous sentence is nocuous or innocuous, based on a set of heuristics that draw on human judgments, which we collected as training data. The tool focuses on coordination ambiguity.

Kamsties et al. [12] described pattern-driven inspection technique to detect ambiguities in requirements.

Mich and Garigliano [13] investigate the use of a set of ambiguity for the measurement in syntactic and semantic ambiguity, which is implemented in tool LOLITA using NLP algorithms.

Kiyavitskaya et al. [14] proposed a two-step approach in which lexical and syntactic analysys was performed to identify ambiguity. An automated tool was implemented to measure what is potentially might be ambiguous specifically for each sentence.

Another important proplem related to specification quality assesment is a choice of correct criteria for overall evaluation. Davis et al. evaluated 24 criteria and metrics for determination of the overall requirements specification quality [15]. Some of the criteria may affect and even contradict each other. Therefore, the authors made a conclusion that a perfect requirements specification does not exist. Another approach was proposed by Wilson et al. – he counted the occurrences of certain expressions in a document to evaluate its quality [16] with indicators that include completeness and consistency. This group of researchers developed a tool that focus on a broader understanding of requirements quality, instead of just a single aspect. Implemented ARM tool is based on the IEEE 830 standard and aims at developing metrics for requirements quality.

Ambriola and Gervasi [17] developed a web-based NLP tool, called Circe, which was designed to facilitate the gathering, elicitation, selection, and validation of requirements.

Unlike other related works show attempts to evaluate requirements in automatically by only one quality criteria, current paper describes an approach to identify several correct quality attributes with correlated metrics to measure, combining them into one overall evaluation in an automated way. Moreover, all the mentioned-above tools support only English language and don't support requirements written in Russian.

3. GQM approach

The Goal-Question-Metric is a method based on system of questions and simple answers about properties evaluation [18]. This approach consists of three main steps: specifying goals, pointing relevant attributes and providing measurements. GQM framework helped to define appropriate metrics and estimate the quality of requirements in current case. The goal should be defined for an object, with a purpose, from a perspective, in an environment. The overall goal of current project it to measure quality of requirements and it can be formulated by following template:

Analyze requirement quality
for the purpose of improving
with respect to quality attributes
from the viewpoint of project managers
in the context of product development.

In addition, several sub-goals were identified, which should be fulfilled to achieve the main goal. For instance:

Sub-goal: Analyze requirement unambiguity for the purpose of improving with re-spect to quality attributes from the viewpoint of project managers in the context of product development.

Question: How many vague words and weak phrases make requirement ambiguous?

Metric: Number of ambiguous words in 1 requirement divided by an average number of words in 1 requirement.

In this approach, identification of the questions and metrics allows to properly clarify the goals in order to achieve the transparency and propose how and why the goals are supposed to be achieved. Clarification becomes more concrete during the movement from top level to bottom and helps to avoid abstract unreal goals.

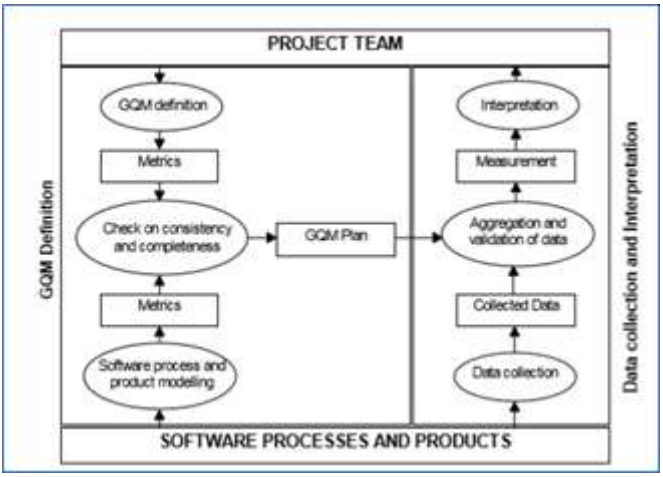

Fig. 2. Phases of Goal/Question/Metric approach [19]

GQM approach is supported by several specific methodological phases (fig. 2) [19].

- Definition – goals, questions, metrics and hypotheses are defined and documented. Main attributes, formulas and measurement approaches and exact metrics are defined.
- Data Collection – searching and counting for ambiguous words and other quality indicators in source text.
- Interpretation – collected data is processed into quality measurement results, that provides answers on defined questions to reach the goal.

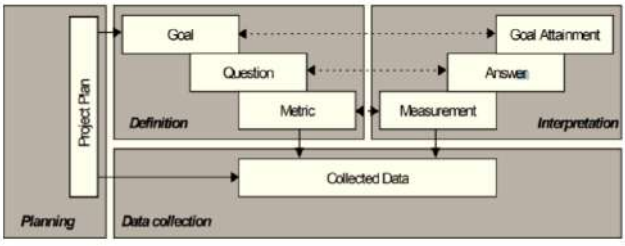

Fig. 3. Process of collecting metrics and measurements [19]

GQM approach in current case consists of 5 main steps (fig. 3) [19].

- Business goal setting. The main purpose of this case is automated evaluating quality of software requirements by range of attributes.
- Generating questions. Breaking down goals into components, defining them in a quantifiable way, e.g., «How much should the proposal structure be complied with so that the requirement has the quality property of completeness»?
- Specifying measures. Detecting metrics that should be collected to answer questions, e.g., «Percentage of matching requirement sentence structure template».
- Defining mechanism of data collection. Measures are collected by semantic and syntax text analysis based on matching words with predefined dictionaries.
- Gathering and analyzing of collected data. Calculated metrics should be interpreted into quality

estimation for each requirement and overall Software Requirements Specification (SRS).

Process of measuring quality in software development has it's certain difficulties. In order to understand the effects of actions that are implemented in software development and gain the understanding of how the improvements can be made for a future process a certain purpose should be put in measurement process. The purpose may be:

1. Understanding of the product requirements. Correct measurements will allow to see the graphical or mathematical representation of a requirement elicitation process, whether it will be a time spend on describing every feature.
2. Controlling the product requirements. While having a graphical representation of the SRS document, the relations between different requirements and user actions can be identified, which further would allow to control the impact on the development process in total.
3. Improving the product requirements. can be achieved after the control of the development processes is gained. Certain improving effect can be applied to processes, variables and their relationships.

   Metrics for the requirements should allow to determine their quality for the current development process and to represent collected resulting data in a graphical way.

4. Quality attributes

Many authors in their methodologies have already defined the key interdependent (Fig. 4) quality attributes [20].

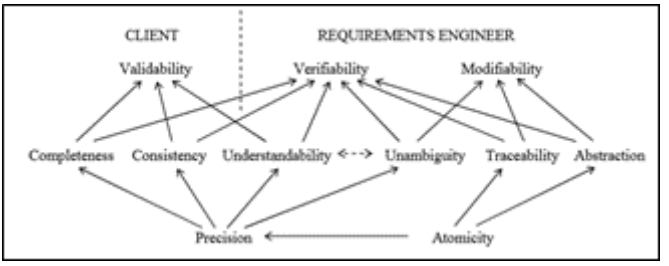

Fig. 4. Dependencies between of qualitative attributes [6]

- Validity – the clients should be able to validate (confirm) the requirement according to their needs.
- Verifiability – the engineer must be able to verify that the system-to-be meets the specified system.
- Modifiability – requirements must be able modifiable with ease for maintenance.
- Completeness – all client's needs must be covered.
- Consistency – should not be any contradiction among requirement.
- Understandability – the requirements are correctly understood without difficulty.
- Unambiguity – there exists only one interpretation of the requirement (no ambiguous words in the requirement sentence).
- Traceability – there exists an explicit relationship of each requirement with design, implementation and testing artefacts.
- Singularity – each requirement is clearly determined and identified, without mixing it with other requirements.

Mentioned-above attributes come out from following types of unbounded or ambiguous terms that should be avoided according to the standard:

- superlatives ('best', 'most');
- subjective language ('user friendly', 'easy to use', 'cost-effective');
- vague pronouns ('it', 'this', 'that');

- ambiguous terms such as adverbs and adjectives (‘almost always’, ‘significant’, ‘minimal’) and ambiguous logical statements (‘or’, ‘and/or’);
- open-ended, non-verifiable terms (such as ‘provide support’, ‘but not limited to’, ‘as a minimum’);
- comparative phrases (‘better than’, ‘higher quality’);
- loopholes (‘if possible’, ‘as appropriate’, ‘as applicable’);
- terms that imply totality (‘all’, ‘always’, ‘never’, and ‘every’);

Unambiguity. It requires that only one semantic interpretation of the requirement exists. To evaluate the ambiguity of each requirement, we propose to use dictionaries with a set of words, which indicates ambiguity in the requirement [21][22]. There are several types of them (Table 1).

- Connection words dictionary – (“and”, “or”, “but”, “however”, “otherwise”, “even”, “although”, etc.) the usage of such words not a problem in itself, but their too frequent use leads to a decrease in the quality of requirements, especially in terms of uniqueness and ambiguity.
- Negative adverbs dictionary – the negative particle is the word not used to indicate negation, denial, refusal, or prohibition. Repeated use of such words makes a sentence difficult to understand and decrease the ambiguity of the requirement.
- Anaphoric expressions dictionary – the use of expressions, the interpretation of which depends on other expressions previously encountered in the text, for example: “which”, “he”, “she”, “it”, “they”, “where”, “this”, “that”, etc. Requirements containing anaphora usually do not have characteristics of clarity and unambiguity.
- Undefined terms dictionary – In addition to connective words, the quality of requirements is significantly affected by the use of vague terms that lead to ambiguity.

Table 1. Types of dictionaries

Type	**English examples**	**Russian examples**
Quality	‘best’, ‘most’, ‘appropriate’, ‘adequate’	‘лучший’, ‘самый’, ‘подходящий’, ‘адекватный’
Subjectivity	‘user-friendly’, ‘easy to use’, ‘rational’	‘легкий в использовании’
Quantity	‘about’, ‘significant’, ‘minimal’, ‘few’, 'all', 'each', 'every', 'any', 'few', 'little', 'many', 'much', 'several', 'some'	‘примерно’, ‘несколько’, ’немного’
Frequency	‘almost always’, ‘usually’, ‘as a rule’, ‘never’	‘почти всегда’, ‘обычно’, ‘как правило’
Persistency	‘long’, ‘longer’, ‘durable’, ‘momentary’	‘долго’, ‘больше’, ‘прочный’, ‘сиюминутного’
Probability	‘probably’, ‘possibly’, ‘if it possible’, ‘unlikely’	‘возможно’, 'если это возможно’, 'вряд ли'
Open listings	‘etc.’, ‘and so forth’, ‘and so on’	‘и так далее’
Position / size	‘close’, ‘bigger’, ‘tall’, ‘far’, ‘short’, ‘small’, ‘huge’	‘близко’, ‘больше’, ‘падение’, ‘далеко’, ‘короткий’, ‘небольшой’, ‘огромный’
Comparative phrases	‘better than’, ‘higher quality’, ‘same as’	‘лучше, чем’, 'выше качество’, 'то же, что’
Loopholes	‘if possible’, ‘as appropriate’, ‘as applicable’	‘если возможно’, ‘соответствующие’, ‘в качестве применимого’

Weak adverb or adjective	'as desired', 'at last', 'either', 'eventually', 'if appropriate', 'if desired', 'in case of', 'if necessary'	'по желанию', 'наконец', 'либо', 'в конечном счете', 'если уместно', 'если желательно', 'в случае', 'если необходимо'

As the metric for assessing ambiguity, was used the following formula:

$$Unambiguity\ \% = \left(1 - \frac{N_{ambig}}{N_{total}}\right) \times 100$$

Where N_{ambig} – the number of words in the requirement, N_{ambig} – the number of ambiguous words in the requirement.

Singularity. Statement of the requirement must relate to only one unique requirement that does not overlap with others. The presence of several modal words tells us that the requirement contains several meanings and that the statement does not have the characteristic of singularity. These words may include could, may, might, can, should, will, shall, must, would, etc. The number of connective words may also indicate the presence of several requirements within one (mentioned above). As the metric for assessing singularity, was used the following formula:

$$Singularity\ \% = \left(1 - \frac{(N_{modal} - 1) + N_{connective}}{N_{total}}\right) \times 100$$

where N_{total} – the number of words in the requirement, N_{modal} – the number of modal verbs which are not zero, $N_{connective}$– the number of connective words in the requirement.

Subjectivity. This attribute indicates the presence of perspectives, feelings, or opinions entering the decision-making process. The leading causes of subjectivity in requirements can be:

- dangerous plural with ambiguous reference,
- combination of “and” and “or” that leads to unclear associativity,
- unclear inclusion,
- passive voice,
- imprecise and inside behavior,
- negative or too broad reason.

Readability. This attribute indicates how easily requirement text can be read and understood, it can be based on the number of syllables per word and number of words per sentence. It can be calculated by Flesch-Kincaid Grade Level [23], Coleman-Liau Grade Level [24], and Smog Grade [25]. The second one was chosen:

$$Redabiliti\ CLI = 0.0588\ L - 0.296S - 15.8$$

where L – average number of letters per 100 words, S – average number of sentences per 100 words. If CLI is around 10, text is easy to read, but if CLI > 15 text is too difficult for understanding. A mapping into percentage interpretation was made (if CLI index is more than 17.5, than readability is 0%) by following formula:

$$Readability\ \% = \left(1 - \frac{|\ CLI - 12.5\ |}{5}\right) \times 100$$

Completeness. It requires that the requirement contain all necessary elements, including constraints and conditions, to enable the requirement to be implemented [7].

Example of structure template in Table 2 for this requirement: «In the Combat Zone, an HQ Switch, which is identical to a trunk node switch, shall be given two independent links to at least two other nodes in the network».

Table. 2. Structural template for completeness

Element	**Text**
Actor	an HQ switch
Conditions of Action Action	in the Combat Zone two independent links

Object of action	-
Constraints of action	To at least two other nodes in the net
Source of Object	To at least two other nodes in the net
Destination of Action	Which identical to a trunk node switch

Completeness quality attribute was calculated by this formula:

$$Completeness\ \% = \frac{N_{filled}}{N_{total}} \times 100$$

where N_{total} – the number of elements in the structural template, N_{filled} – the number of elements from template that were identified in requirement sentence.3

5. Natural language Processing

Natural Language Processing is a field of computer science and computational linguistics that aims to analyze linguistic data from input using computational methods and techniques [26]. The natural language is very complicated, it is subject to syntactic and semantic rules. It studies the conceptual dimension that refers to «pragmatic» actions which are intended. The syntactic rules describe the major pattern of a sentence such as nouns, adjectives, and verbs [20]. The semantic rules refer to the meaning of each word in the sentence and relation between words when they are combined, which is «compositional semantic» [27].

Natural language texts are used to be analyzed in a sequential process. This process starts with lexical and structural elements. For its purpose text should be parsed in a search of the most suitable syntax tree. After that some complex techniques are applied for interpretation of the semantic content due to meaning understanding. Of course, such analysis does not allow to understand fully the content and get independent meaning of the sentence without any discrepancies.

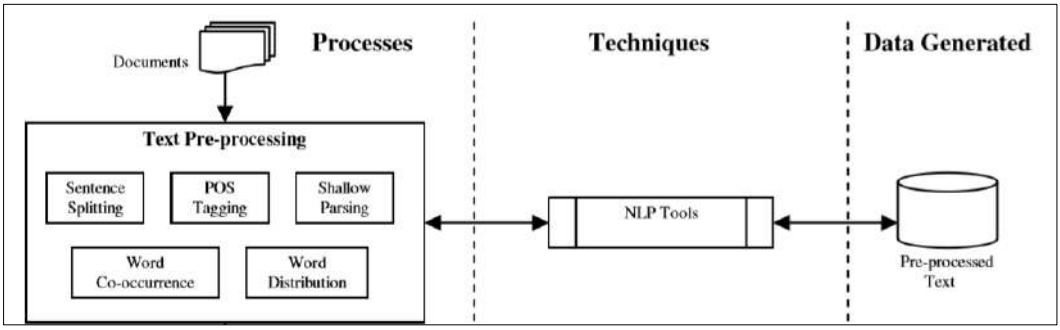

Fig. 5. NLP workflow for requirement analysis

Several techniques were used in current model workflow: splitting sentence in syntax tree, part-of-speech tagging, morphological analysis and calculationg word distribution and co-occurance by redefined dictionaries (Fig. 5). Inputed text document with requirements should go through several text preprocessing steps: sentence splitting, POS-tagging, and phrase-based shallow parsing.

Sentence splitting. At first, the text is splitted into a set of sentences by using a sentence boundary detector.

POS-tagging. Then, for each requirement sentence, the parser based on individual words and associated phrase information that used to obtain word lemma and POS tags such as noun, verb, adjective, adverb, etc. In current model the Stanford NLP library [28] was used for this. POS tagging helped to determine so-called substituting pronouns. A detailed description of POS tagging technical details is beyond the scope of this paper, but can be found, for example, in [29]. Given a sentence in natural language text, it determines the role and function of each single word in the sentence. The output is usually a so-called tag for each word, e.g. whether a word is an adjective, a particle or a possessive pronoun. These are pronouns that do not repeat the original noun and, thus,

need a human's interpretation of its dependency. A syntax tree shows the main structure of the sentence (Fig. 6), where tree's leafs are the words of the sentence and inner nodes express the sentence's composition. In example «the channel selection» forms a nominal phrase (NP), as indicated by their common parent node NP. The additional information «of the headphones» is added as prepositional phrase (PP). The noun phrase and the prepositional phrase form a new nominal phrase, which is the object of the verb «changes».

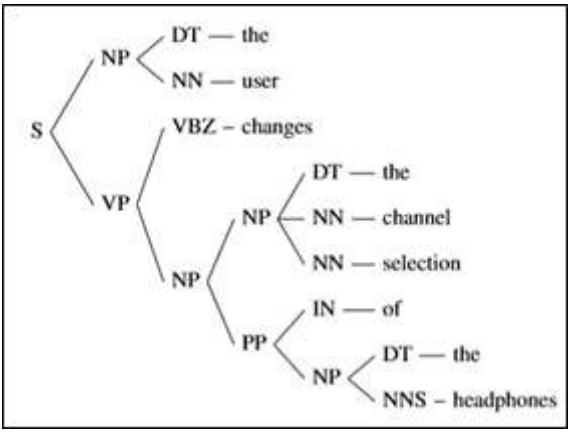

Fig. 6. Syntax tree for NLP workflow

Morphological Analysis. Based on POS-tagging more detailed analysis of text was performed that determines its inflection. This step contains identifying a verb's tense or an adjective's comparison. The main outcome of this step is analysis for usage of adverbs and adjectives in their comparative or superlative form.

Dictionaries. For describing different quality attributes were used several dictionatries with ambigious phrases and words based on quality standards and case study experience. Normalisation technique for dicitionry words called lemmatisation was applied, that reproduces the original form of a word. This technique is very similar to stemming, Porter Algorithm [30], that based on the POS tag as the word's morphological form instead of heuristics.

6. Prototype

To fully support the extraction of metrics for all before-mentioned quality attributes, the prototype should have several features [Fig. 7].

The prototype is a software tool which main goal is to perform requirements quality measurement. Requirements can be of any type expressed in the text form: functional, non-functional, use-cases. The prototype is able to perform several functions:

- integration with project management system to gather textual requirements from it (via is API).
- perform syntax and semantic analysis of said requirements (supporting Russian language [21][22]).

The core of the prototype is the Requirement Quality Model which contains a consistent set of requirements quality metrics and is expressed in algorithms on how to measure these metrics and how to draw conclusions (average quality of a requirement/set of requirements). The prototype provides a requirement engineer with a graphical user interface or command-line interface to obtain the results of requirements measurement.

For NLP were used custom analogues of Python libraries Wordnet [31] and Spacy [32] with Russian language support. Analysys of ambiguity was implemented based on open-source microservice OpenReqEU.

To get results of the requirements analysis the prototype provides the requirements engineer with the either graphical or command line interface. Here are some of the interface functions that are available:

- list all the requirements;
- show quality metrics of the specific requirement;
- show quality metric for all requirements analysed.

The dictionaries of ambigious words were translated into Russian language. Docker container for OpenReqEU microservice was rebuild and used as external API for further process of quality assessment. All ambigius words in requirements highlighted according to their category after service finished its work.

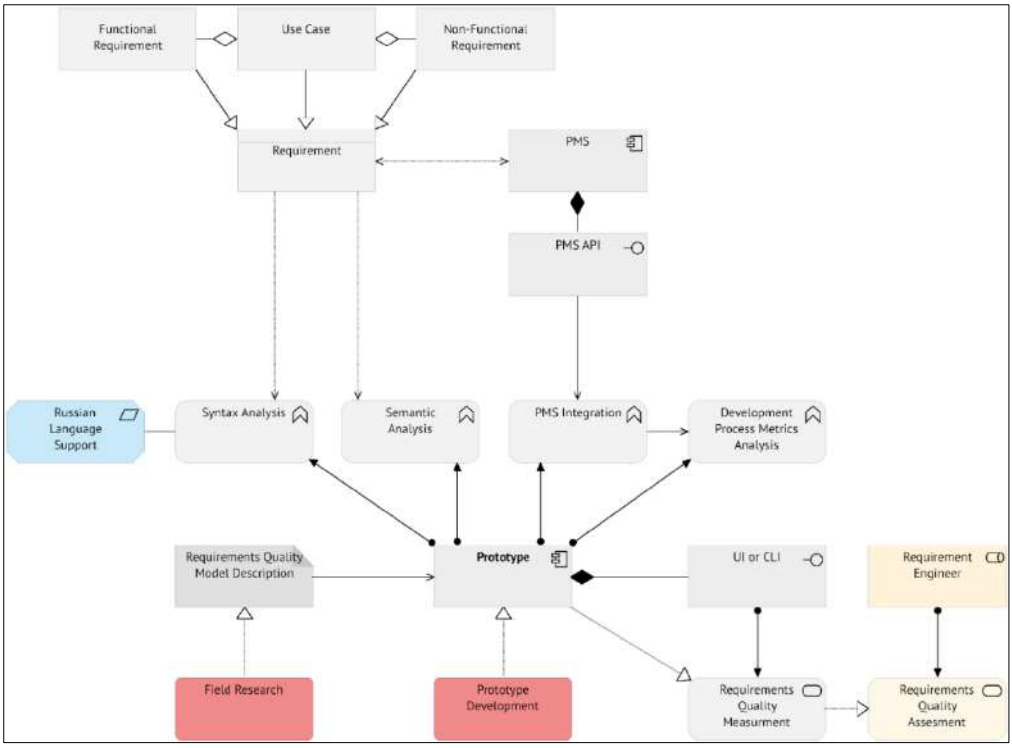

Fig. 7. Prototype scope in ArhiMate [33] notation

On the next step one more external API was used for evaluating readability indexes by service *readability.io* – text of every requirement was uploaded and resulting number was recieved from website.

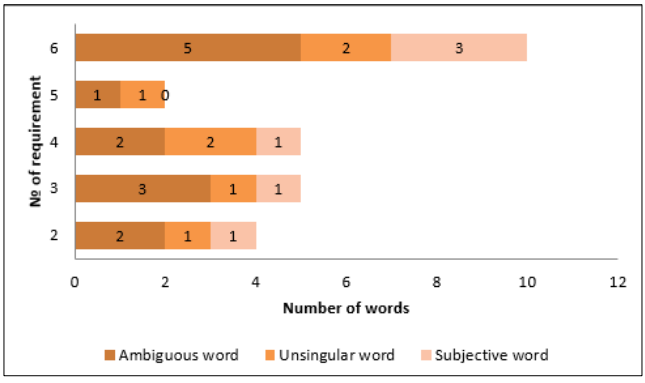

Fig. 8. The number of weak words per requirement

For graphical representation of evaluated quality results Visual Paradigm Diagram was used. Spider graph and stacked histogram were chosen as the most appropriate visualisaition of collected data. All the metrics that were calculated in prototype automatically synchronized with Visual Paradigm Service and published as a web-dashboard.

7. Results

After implementing the proposed solution on requirements, it was tested on the sample requirement text. As a result, the following distribution of weak words shown in Fig. 8 was got. These weak words were highlighted in GUI and classified by different types of ambiguity (Fig. 9).

Fig. 9. Hightlighted ambiguous words in every requirement

Final evaluation about overall quality was made (Fig 10).

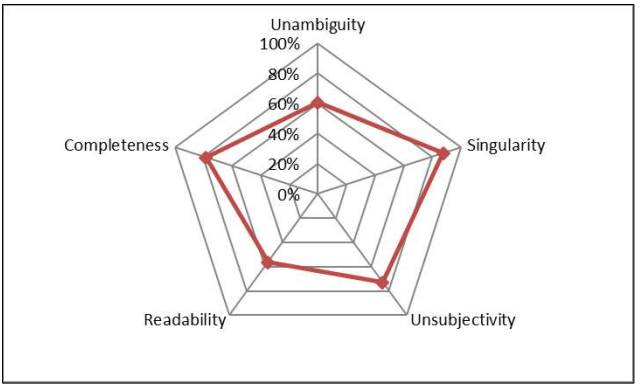

Fig. 10. Overall quality of all requirements by attributes

8. Conclusion and discussion

Natural language still prevails in the majority of requirement documents. Software engineers need ways to cope with the ambiguity inherent in natural language requirements. In order to minimize their side effects at the early stages of the software development lifecycle, it is important to

develop scalable automated solution to detect potential nocuous ambiguities in natural language requirement specifications.

The usage of quality metrics in a software development lifecycle requires considering three important aspects. Firstly, obtaining all mentioned-above measurements by hand would be misleading, therefore automated tools become required. Secondly, an automated prototype implementation should avoid the refusal of requirements engineers – this tool is created due to help in improvement of requirements elicitation process, but not for punishment and identifying failures. Finally, decisions about which attributes and metrics to apply should be wisely and gradually made: «Not everything that can be counted counts, and not everything that counts can be counted» – Albert Enshtein.

Despite the fact that quantitative measurement is one of the foundations of modern empirical science, they should be used with caution and wisdom. Assessing the quality of requirements demands human judgment. This judgment can be assisted, but not replaced, by objective measurements. Automated tool that provides low-level quality indicators can provide valuable hints to improve high-level quality features of requirements.

In this paper an automated approach for characterizing and identifing potentially nocuous ambiguities was described. Given a natural language requirements document, ambiguous instances contained in the sentences were first extracted. Identified ambiguities can be the reason of misunderstanding among different readers. The implementation can be usable by requirements analysts and will allow them to experiment with iterative identification of potential ambiguity moments in requirement documents.

References / Список литературы

[1] ISO/IEC/IEEE 29148:2018 International Standard – Systems and software engineering – Life cycle processes – Requirements engineering.

[2] R. Abbot. Program design by informal English descriptions. Communications of the ACM, vol. 26, no. 11, 1983, pp. 882-894.

[3] H. Femmer et al. Rapid requirements checks with requirements smells: Two case studies. In Proc. of the 1st International Workshop on Rapid Continuous Software Engineering, 2014, pp. 10-19.

[4] Y. Pisan. Extending requirement specifications using analogy. In Proc. of the 22nd International Conference on Software Engineering, 2000, pp. 70–76.

[5] F. Fabbrini et al. The linguistic approach to the natural language requirements quality: Benefit of the use of an automatic tool. In Proc. of the 26th Annual NASA Goddard Software Engineering Workshop, 2001, pp. 97–105.

[6] N. Power. Variety and quality in requirements documentation. In Proc. of the 7th International Workshop on Requirements Engineering: Foundation for Software Quality, 2001, pp. 165–170.

[7] S.J. Körner, M. Landhäußer, and W. F. Tichy. Transferring research into the real world: How to improve RE with AI in the automotive industry. In Proc. of the 1st International Workshop on Artificial Intelligence for Requirements Engineering, 2014, pp. 13-18.

[8] K. Verma and A. Kass. Requirements analysis tool: A tool for automatically analyzing software requirements documents. Lecture Notes in Computer Science, vol. 5318, 2008, pp. 751–763.

[9] L. Goldin and D.M. Berry. Abstfinder, a prototype abstraction finder for natural language text for use in requirements elicitation: design, methodology, and evaluation. In Proc. of the First International Conference on Requirements Engineering, 1994, 18-22.

[10] B.S. Lee and B.R. Bryant. 2004. Automation of software system development using natural language processing and two-level grammar. Lecture Notes in Computer Science, vol. 2941, 2002, pp. 219-233.

[11] H. Yang et al. Automatic detection of nocuous coordination ambiguities in natural language requirements. In Proc. of the IEEE/ACM International Conference on Automated Software Engineering, 2010, pp. 53-62.

[12] E. Kamsties, D. Berry, and B. Paech. Detecting ambiguities in requirements documents using inspections. In Proc. of the First Workshop on Inspection in Software Engineering (WISE'01), 2001, pp. 68-80.

[13] L. Mich and R. Garigliano. Ambiguity measures in requirement engineering. In Proc. of International Conference on Sftware – Theory and Practice (ICS2000), 2000, pp. 39-48.

[14] N. Kiyavitskaya et al. Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. Requirements Engineering Journal, vol. 13, 2008, pp. 207-240.
[15] A. Davis et al. Identifying and measuring quality in a software requirements specification. In Proc. of the 1st International Software Metrics Symposium, 1993, pp. 141–152.
[16] W. Wilson, L. Rosenberg, and L. Hyatt. Automated analysis of requirement specifications. In Proc. of the 19th International Conference on Software Engineering, 1997, pp. 161–171.
[17] V. Ambriola and V. Gervasi. Processing natural language requirements. In Proc. of the 12th international conference on Automated software engineering, 1997, 36-45.
[18] Basili, Victor; Gianluigi Caldiera; H. Dieter Rombach. The Goal Question Metric Approach, NASA GSFC Software Engineering Laboratory, 1994.
[19] Rini Van Solingen, Egon Berghout. The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development. McGraw-Hill Education, 1998, 280 p.
[20] Gonzalo Génova et al. A framework to measure and improve the quality of textual requirements. Requirements Engineering, vol. 18, 2013, pp. 25-41.
[21] K.I. Gaydamaka. Characteristics and quality indicators of requirements for the Russian-speaking engineering environment. In Proc. of the Conference «Technologies for the development of information systems», TRIS-2017, 2017, pp. 80-85 (in Russian) / К.И. Гайдамака. Характеристики и индикаторы качества требований для русскоязычной инженерной среды. Труды конференции «Технологии разработки информационных систем», ТРИС-2017, 2017, стр. 80-85.
[22] V.K. Batovrin, K.I. Gaydamaka. Some special aspects of assessing the characteristics of system requirements. Informatization and communication, no. 4, 2017, pp. 191-196 (in Russian) / В.К. Батоврин. К.И. Гайдамака. Некоторые особенности оценки характеристик требований к системам. Информатизация и связь, no. 4, 2017, pp. 191–196.
[23] J.P. Kincaid et al. Derivation of new readability formulas (automated readability index, fog count, and flesch reading ease formula) for Navy enlisted personnel. Research Branch Report 8–75. Chief of Naval Technical Training: Naval Air Station Memphis, 1975.
[24] Meri Coleman and T.L. Liau. A computer readability formula designed for ma-chine scoring. Journal of Applied Psychology, vol. 60, 1975, pp. 283–284.
[25] G. Harry McLaughlin. SMOG Grading – a New Readability Formula (PDF). Journal of Reading, vol. 12, no. 8, 1969, pp. 639–646.
[26] Anderson Rossanez. Semi-Automatic Checklist-Based Quality Assessment of Natural Language Requirements. Master Thesis, University of Campinas, Brazil, 2017, 96 p.
[27] Daniel Jurafsky & James H. Martin. Speech and Language Processing: An introduction to natural language processing, computational linguistics, and speech recognition. Prentice Hall, 2000, 934 p.
[28] D. Jurafsky and J. H. Martin. Speech and Language Processing. Prentice Hall, 2008, 1032 p.
[29] K. Toutanova et al. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), 2003, pp. 173-180.
[30] M. Porter. An algorithm for suffix stripping. Program: electronic library and information systems, vol. 14, no. 3, 1980, pp. 130-137
[31] ru-wordnet. GitHub repository, 2017. Available at: https://github.com/jamsic/ru-wor. Accessed 20 Nov. 2019.
[32] Y. Baburov. spacy-ru. GitHub repository, 2018. Available at: https://github.com/buriy/spacy-ru. Accessed 20 Nov. 2019
[33] Pubs.opengroup.org. ArchiMate 3.1 Specification. Available at: https://pubs.opengroup.org/architecture/archimate3-doc/. Accessed 20 Nov. 2019.

Information about the author / Информация об авторе

Evgeny Valerievich TIMOSHCHUK. Master student at the Institute of Computer Science and Technology. Scientific interests: requirements management, quality assessment of requirements, natural language processing.

Евгений Валерьевич ТИМОЩУК. Студент магистратуры Института компьютерных наук и технологий. Научные интересы: управление требованиями, оценка качества требований, обработка естественного языка.

DOI: 10.15514/ISPRAS-2020-32(2)-3

Application of machine learning technology to analyze the probability of winning a tender for a project

N.B. Kultin, ORCID 0000-0002-7967-6542 <kultin_nb@spbstu.ru>
D.N. Kultin, ORCID 0000-0003-3010-150X <kultin.dn@edu.spb.stu.ru>
R.V. Bauer, ORCID 0000-0002-4703-3833 <bauer.rv@edu.spb.stu.ru>
Peter the Great Saint-Petersburg Polytechnic University,
29, Polytechnicheskaya, Saint-Petersburg, 195251, Russia

Abstract. The possibility of using machine learning technology to solve the problem of project analysis in order to support the decision to participate in the tender for the implementation of the project is substantiated and shown using a specific example. The approaches are described and the process of solving the problem of binary classification of projects using libraries of the Python language is shown. Attention is paid to the problem of choosing an algorithm for constructing a membership function, the problem of generating and analyzing input data, and evaluating the accuracy of a solution. It is shown that for the considered problem the best solution is provided by the logistic regression algorithm.

Keywords: machine learning; artificial intelligence; project risk analysis; project management; tender for the implementation of the project

For citation: Kultin N.B., Kultin D.N., Bauer R.V. Application of machine learning technology to analyze the probability of winning a tender for a project. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 29-36. DOI: 10.15514/ISPRAS–2020–32(2)–3

Применение технологии машинного обучения для анализа вероятности выигрыша тендера на выполнение проекта

Н.Б. Культин, ORCID 0000-0002-7967-6542 <kultin_nb@spbstu.ru>
Д.Н. Культин, ORCID 0000-0003-3010-150X <kultin.dn@edu.spb.stu.ru>
Р.В. Бауэр, ORCID 0000-0002-4703-3833 <bauer.rv@edu.spb.stu.ru>
Санкт-Петербургский политехнический университет Петра Великого,
Россия, 195251, Санкт-Петербург, ул. Политехническая, д. 29

Аннотация. Обоснована и показана на конкретном примере возможность применения технологии машинного обучения для решения задачи анализа проектов с целью поддержки принятия решения об участии в тендере на выполнение проекта. Изложены подходы и показан процесс решения задачи бинарной классификации проектов с использованием библиотек языка Python. Уделено внимание проблеме выбора алгоритма построения функции принадлежности, задаче формирования и анализа исходных данных, оценки точности решения. Показано, что для рассматриваемой задачи наилучшее решение обеспечивает алгоритм логистической регрессии.

Ключевые слова: машинное обучение; искусственный интеллект; анализ рисков проекта; управление проектами; тендер на выполнение проекта

Для цитирования: Культин Н.Б., Культин Д.Н., Бауэр Р.В. Применение технологии машинного обучения для анализа вероятности выигрыша тендера на выполнение проекта. Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 29-36 (на английском языке). DOI: 10.15514/ISPRAS–2020–32(2)–3

1. Introduction

In modern conditions, many enterprises in various industries are guided by the implementation of projects whose customer is the «state». In accordance with the legislation of the Russian Federation, the executors of such projects are selected based on the results of open tenders. Participation in the tender implies a significant amount of preparatory work related to the preparation of tender documentation, including the development of a technical and economic proposal. Since the result of the tender may be a loss (failure to receive an order for the project), the costs associated with preparing for the tender can be considered as risky. It is possible to reduce the likelihood of losses associated with losing a tender by «sifting» projects at the first stage of preparation for participation in the tender by analyzing them in detail before making the final decision to participate in the tender and starting work on the development of a feasibility study. It is possible to realize the «sifting» of projects on the basis of machine learning technology, solving the classification problem. As a result of solving the classification problem, the analyzed project can be classified as "prospective", in this case, it is advisable to continue the preparation for bidding, or «unpromising».

The purpose of this study is to justify the possibility of using machine learning technology to solve the problem of analyzing projects with the aim of deciding whether to participate in a tender for its implementation, to develop an algorithm for classifying projects to assess the possibility of obtaining a contract for a project.

Materials and research methods. To assess the possibility of obtaining a contract for an engineering project, most companies use qualitative risk analysis methods. In practice, the most widely used method of expert assessments and the method of analogy [1]. The advantages and disadvantages of these methods are shown in Table 1.

Table 1. Qualitative risk analysis methods of projects

Method	Advantages	Disadvantages
Expert assessment method	Use of reliable knowledge Simple calculations	Limited expert pool Subjective ratings
Method of analogy	Use of reliable knowledge Using knowledge inside the company	Low forecast accuracy Lack of consideration of individual project features

Conducting a quantitative risk assessment of the pre-project stage of work requires significant costs. In order to reduce the cost of assessing the risks of the pre-project stage of work, it is proposed to assess the likelihood of obtaining a contract for the implementation of the project by solving the binary classification problem.

To solve the binary classification problem, machine learning methods will be used. Based on the existing characteristics of the projects, the algorithm will construct a separating hyperplane [2] between projects of the classes «prospective» (the probability of obtaining a contract is high) and «unpromising» (the probability of obtaining a contract is low).

To achieve this goal, it is necessary to solve the following tasks:

- perform statistical analysis of data on the characteristics of projects;

- based on the results of the analysis, select an adequate family of algorithms;
- from the family of algorithms based on the quality criterion, select the best algorithm;
- using training and test data samples, determine the quality indicator of the algorithm.

2. Project attributes

A large number of factors influence the fact of obtaining a contract for the implementation of the project. Information about the degree of influence of each factor is recorded in the vector of project attributes. Each component of the vector is a sign that describes the project, as well as the relationship of the customer and the contractor in the framework of this project. Thus, the vector of project attributes is a description of the pre-project stage of work.

Each attribute of the project must be associated with a numerical value and put it in the corresponding component of the vector. However, not all project characteristics can be expressed numerically. There are categorical (one value from a finite set), binary (value 1 or 0) and other complex types (date, time, geographical location). Complex types can be represented by a set of simple features with a numerical representation [3]. To bring categorical features to numerical representation, the binarization method was used [4]. Each value from the set of values of a categorical attribute is associated with its own attribute. For example, the set of «Upcoming project work» is associated with the signs «Start-up and adjustment work», «Design and survey work», etc. If the type of work corresponding to the characteristic needs to be performed within the framework of the project, then the characteristic takes the value «1», otherwise «0». Methods for converting features are presented in Table 2.

Table 2. Methods for converting of attributes

Characteristic category	**Characteristic name**	**Influence factor**	**Representation type**	**Numeric presentation method**
Attributes of the project	Preliminary budget (budget)	Preliminary assessment of income from the implementation of project	Numerical	Value of the budget specified by the customer
	Type of work (kw _...)	Assessment of the complexity of work on the project	Categorical	Binarization
	Preliminary project implementation period (rz_date)	Duration estimation	Date and time	The number of days between pre-specified project start and end dates
	Place of implementation (rz_place)	Assessment of travel expenses, climate, infrastructure	Geographic	Latitude and longitude (GPS coordinates). It is represented by two real values
Characteristics of the executing company	Project manager (app_manager)	Evaluation of the impact of the compiled project description on further work	Numeric	Number of contracts for applications out of accepted by the manager / Number of all applications accepted by him
	Sales Manager (sale_manager)	Evaluation of work experience	Numeric	Number of applications that the manager in the company

				conducted
	Responsible Manager (res_manager)	Assessment of experience and specialization of the manager	Categorical	Binarization
Characteristics of the customer's company	Client's office (cl_office)	Estimation of expenses for business trips	Numerical	Distance to the client's office in kilometers
	The legal form of the client's company (kc _...)	Assessment of the complexity of interaction between companies	Categorical	Binarization
Other attributes	Collaboration efficiency (coop_eff)	Assessment of the impact of past experience in collaboration	Numeric	Percentage of implemented joint projects from all attempts to work with a specified customer
	Application submission date (app_date)	Estimation of the urgency of the project	Date/time	The number of days between the date of submission of the application and the preliminary start date of the project
Target attribute	Contract conclusion (is_proj)	Whether the contract for the implementation of the project was concluded	Binary	Takes the value "1" if the contract was concluded, "0" if it was not

The set of vectors corresponding to the projects is a matrix, the rows of which are the sets of values of the characteristics of a particular project, and the columns are the sets of values of a specific attribute in all projects of an engineering company [3, 4].

3. Statistical analysis of attributes

In order to assess the quality of the source data and make sure that they can be used to assess the possibility of obtaining a contract for the implementation of the project, it is necessary to perform a statistical analysis of the characteristics. For convenient presentation of the matrix of traits and subsequent statistical studies of individual traits, the Pandas, Numpy, Seaborn libraries of the Python language were used [3].

To obtain a reliable picture, it is necessary to exclude duplication of information in the source data. To solve this problem, it is necessary to construct a matrix of pair correlations of attributes. If the value of the correlation coefficient of attributes modulo more than 0.4, it is concluded that the information in these signs is duplicated and one of the columns of the matrix should be deleted [5]. After this, it is necessary to evaluate the type of data distribution in individual features and the presence of anomalous values. Fig. 1 shows histograms of the distribution of features. The given assessment is taken into account when choosing the parameters of the algorithm for training.

It is also necessary to normalize all numerical signs so that when using these data in training, the relative weights of the signs are not biased. Normalization is performed according to the formula $x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$ [4].

3.1 Building a learning algorithm

After selecting and analyzing the characteristics, we obtain a certain set X and its subset X^l. This subset X^l is the set of all known pairs $(x_i, y_i)_{i=1}^{l}$, where x_i- is a characteristic description of a specific project of the company from the set of projects X, and y_i- is one of two values of the target characteristic from the set of possible responses $Y = \{0, 1\}$, corresponding to the response on the object x_i [6].

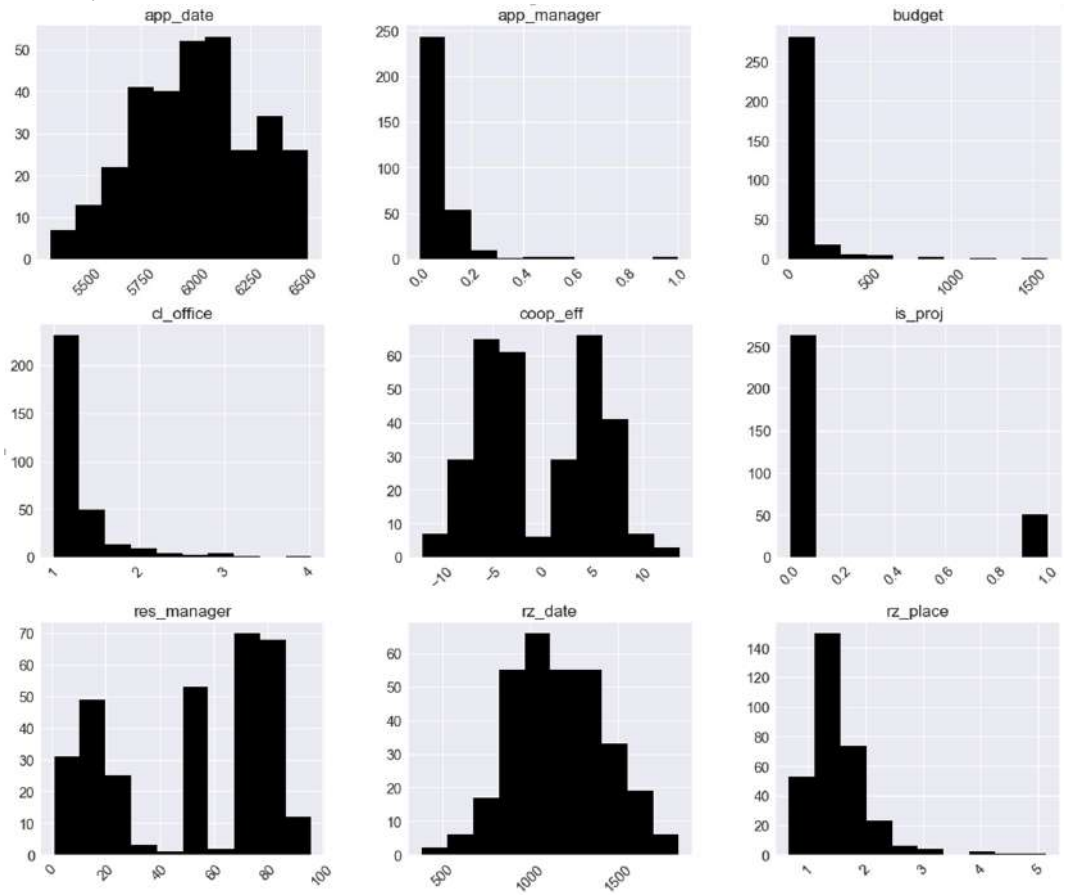

Fig. 1. Attribute value distribution charts

There is a target function $y^*: X \to Y$, that must be restored from known values on the set X^l. The task of learning the algorithm is to build a decisive function $a: X \to Y$, that would approximate the objective function $y^*(x)$, and not only on objects of the set X^l, but on the whole set X [6].

To apply machine learning algorithms, we divide the set X into three subsets: a training set, a test set, and objects without answers. Elements of the training set will be used to solve the learning problem and are elements of the subset X^l but do not participate in the learning of the algorithm. They help identify the problem of retraining. Retraining refers to a situation where the quality of the algorithm on data that did not participate in training drops sharply. To avoid the problem of retraining, the subset X^l needs to be divided into training and test samples so that the ratio of the values of the target attribute in the objects of both samples is the same.

When analyzing the literature on the research topic, it was revealed that the set task is well solved by logistic regression algorithms and support vector methods [7, 8]. The implementation of these algorithms was taken from the ScikitLearn library of the Python language.

To obtain the decisive function a, we will minimize the error functional Q on the training set of 295 vectors. The type of error functional for logistic regression and the support vector method are presented in Table 3 [3, 6, 9].

Table 3. Error functional of selected algorithms

	Logistic regression method	**Support vector method**
Error functional	$\frac{1}{2}ww^T + C\sum_{j=1}^{n}\log(\exp\left(-y_i\left(x_i^T w + b\right)\right) + 1)$, x_i- algorithm response, y_i- valid answer, w - weights vector, b - distance between dividing plane and origin, C - penalty for total error, includes L2 regularization	$\frac{1}{2}ww^T + C\sum_{i=1}^{n}\zeta_i$, on condition $y_i(w^T\phi(x_i) + b) \geq 1 - \zeta_i$. x_i, y_i, w, b и C have the same meaning as in logistic regression, ζ_i- error value at each object.

Three metrics were used to evaluate the algorithms: accuracy (T), completeness (P), f-measure (F). Accuracy shows the proportion of class objects correctly classified by the decisive function among all objects that the function has assigned to this class. Completeness shows the proportion of class objects correctly classified by the decisive function among all objects of this class in the test sample. The F-measure is the harmonic mean between accuracy and completeness. The values of the quality metrics of the algorithms in the test set are presented in Table 4.

Table 4. Comparison of algorithm quality metrics

	Logistic regression method			**Support vector method (polynomial loss function)**			**Number of vectors**
Metrics	T	P	F	T	P	F	
Value for class "1"	1.00	0.73	0.84	0.57	0.73	0.64	11
Value for class "0"	0.97	1.00	0.98	0.96	0.93	0.95	84
Average	0.97	0.97	0.97	0.92	0.91	0.91	95

3.2 Evaluation of the quality of the algorithm

To obtain a more objective assessment of the quality of the algorithms, cross-validation was conducted on 7 sets of test and training samples [10]. During the cross-validation process, the entire data set (matrix X^l) is divided into several parts - in this case, seven. Next, one part of the data becomes a test sample, and the algorithm is trained on the remaining six. Then, accuracy is evaluated on the test sample and the result is recorded. Then the next of seven pieces of data becomes a test sample. In such an artificial way, the algorithm is checked for adaptability to data from different sets [3, 4]. The cross-validation results for the algorithms are shown in Table 5.

Table 5. Cross validation results

Quality ratings	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Logistic regression method	0.95	0.93	0.95	0.95	0.90	0.86	0.88
Support vector method	0.91	0.93	0.95	0.95	0.93	0.86	0.86

Analysis of the data shows that the logistic regression algorithm is more suitable for a given data structure and better solves the problem.

The algorithm was tested on data on 53 projects, applications for which were received by the project department of the enterprise and were being processed at the time when the algorithm

evaluated them using known parameters. After receiving information on the results of the work on these applications, quality metrics for evaluating the algorithm were measured. The measurement results are shown in Table 6.

Table 6. The result of testing the algorithm on real data

Metrics	T	P	F	Number of vectors
Value for class “1”	0.094	1.000	0.172	4
Value for class “0”	1.000	0.408	0.508	49

The test results show that the algorithm coped quite well with the solution of the problem, since all projects classified by the algorithm as unpromising are indeed such. At the same time, the algorithm correctly classified only 40% of unpromising projects. The algorithm has not missed a single truly promising project.

The probabilities of belonging to classes were also analyzed, which the algorithm evaluated when working on test data. It was found that these estimates are very high, which indicates excessive «confidence» of the algorithm.

4. Conclusions

1. Using machine learning technology to build a binary classification algorithm for projects is possible, but requires a significant amount of information about the characteristics of projects and the results of previous tenders.
2. The constructed classification algorithm can be adapted for use in any companies engaged in project activities. Using the algorithm will allow you to rank current projects, reduce the cost of the pre-project stage.
3. In the future, it is possible to refine the algorithm in the direction of expanding the number of features, increasing the accuracy of tuning the hyperparameters of the algorithm, which will make it possible to obtain more accurate and reasonable estimates, reduce the degree of confidence of the algorithm in assessing the probability of belonging to the class, and increase the accuracy metric for the class of promising projects while maintaining the current completeness value.

References / Список литературы

[1] Konstantinov G.N. Strategic management: concepts. Moscow, Business-alignment, 2009, 239 p. (in Russian) / Константинов Г.Н. Стратегический менеджмент: концепции. — М.: Бизнес-элайнмент, 2009, 239 стр.

[2] Principles of Machine Learning. Microsoft, 2018. Available at: https://www.edx.org/course/principles-machine-learning-microsoft-dat203-2x-6.

[3] Training on tagged data. Moscow Institute of Physics and Technology, Yandex School of Data Analysis. Coursera, 2018. Available at: https://www.coursera.org/learn/supervised-learning (in Russian) / Курс “Обучение на размеченных данных”. МФТИ, Яндекс.

[4] Machine Learning. Yandex School of Data Analysis, Moscow Institute of Physics and Technology, 2018. Available at: http://lectoriy.mipt.ru/course/MachineLearning-L (in Russian) / Курс “Машинное обучение”. Яндекс, МФТИ, 2018.

[5] Mathematics and Python for data analysis. Moscow Institute of Physics and Technology, Yandex School of Data Analysis. Coursera. 2018. Available at: https://www.coursera.org/learn/mathematics-and-python (in Russian) / Курс “Математика и Python для анализа данных”. Яндекс, МФТИ, 2018.

[6] Vorontsov K.V. Mathematical teaching methods on precedents (machine learning theory), 2018. (in Russian) Available at: http://www.machinelearning.rU/wiki/images/6/6d/Voron-ML-1.pdf / Воронцов К. Математические методы обучения по прецедентам (теория обучения машин), 2018.

[7] Min-Yuan Cheng, Nhat-Duc Hoang. Interval estimation of construction cost at completion using least squares support vector machine, Journal of Civil Engineering and Management, vol. 20, no. 2, 2014, pp. 223-236.

[8] Min-Yuan Cheng, Nhat-Duc Hoang. Dynamic prediction of project success using evolutionary support vector machine inference model. In Proc. from the 25th International Symposium on Automation and Robotics in Construction, 2008, pp 452-458.

[9] ScikitLearn, Scikit-learn developers (BSD License), 2007 - 2017. Available at: http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression.

[10] Kaftannikov I.L., Parasich A.V. Problems of Training Set's Formation in Machine Learning Tasks. Bulletin of the South Ural State University. Series Computer Technologies, Automatic Control, Radio Electronics, vol. 16, no. 3, 2016, pp. 15–24. (in Russian) / Кафтанников И.Л., А.В. Парасич. Проблемы формирования обучающей выборки в задачах машинного обучения. Вестник ЮУрГУ. Серия Компьютерные технологии, управление, радиоэлектроника, том 16, no. 3, 2016 г., стр. 15-24.

Information about authors / Информация об авторах

Nikita Borisovich KULTIN – candidate of technical sciences, associate professor of the department of management. Research interests: theory and practice of innovative project management, the use of expert systems and machine learning systems to solve management problems in technical and socio-technical systems

Никита Борисович КУЛЬТИН – кандидат технических наук, доцент кафедры управления. Научные интересы: теория и практика управления инновационными проектами, применение экспертных систем и систем машинного обучения для решения задач управления в технических и социо-технических системах.

Данила Никитич КУЛЬТИН – аспирант Института информационных технологий и управления СПбПУ. Научные интересы: управление инновационными проектами, применение теории игр в управлении проектами, портфельное управление

Danila Nikitich KULTIN is a PhD student at the Institute of Information Technology and Management of St. Petersburg Polytechnic University. Research interests: innovation project management, application of game theory in project management, portfolio management

Roman Vladimirovich BAUER is a graduate student. Research interests: innovation project management, machine learning systems in project management.

Роман Владимирович БАУЭР – студент магистратуры. Научные интересы: управление инновационными проектами, системы машинного обучения в управлении проектами.

DOI: 10.15514/ISPRAS-2020-32(2)-4

Comparative Analysis of Homomorphic Encryption Algorithms Based on Learning with Errors

[1,2] *M.G. Babenko, ORCID: 0000-0001-7066-0061 <mgbabenko@ncfu.ru>*
[1] *E.I. Golimblevskaia, ORCID: 0000-0001-7188-8711 <elena.golimblevskaya@gmail.com>*
[1] *E.M. Shiriaev, ORCID: 0000-0002-2359-1291 <ea_or@list.ru>*
[1] *North-Caucasus Federal University,*
1, Pushkin st., Stavropol, 355017, Russia
[2] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. The widespread use of cloud technology allows optimizing the economic costs of maintaining the IT infrastructure of enterprises, but this increases the likelihood of theft of confidential data. One of the mechanisms to protect data from theft is cryptography. Using the classical primitives of symmetric and asymmetric encryption does not allow processing data in encrypted form. Homomorphic encryption is used for processing confidential data. Homomorphic encryption allows performing of arithmetic operations over encrypted text and obtaining an encrypted result that corresponds to the result of operations performed over plaintext. One of the perspective directions for constructing homomorphic ciphers is homomorphic ciphers based on Learning with Errors. In this paper we examine the cryptographic properties of existing homomorphic ciphers (CKKS, BFV) based on Learning with Errors, compare their technical characteristics: cryptographic strength and data redundancy, data encoding and decoding speed, speed of arithmetic operations, such as addition and multiplication, KeySwitching operation speed.

Keywords: Homomorphic encryption; fully homomorphic encryption scheme; Residue Number System; BFV scheme; CKKS scheme; LattiGo; GoLang

For citation: Babenko M.G., Golimblevskaia E.I., Shiriaev E.M. Comparative Analysis of Homomorphic Encryption Algorithms Based on Learning with Errors. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 37-52. DOI: 10.15514/ISPRAS–2020–32(2)–4

Acknowledgements. The reported study was funded by Russian Science Foundation, project number 19-71-10033.

Сравнительный анализ алгоритмов гомоморфного шифрования на основе обучения с ошибками

[1,2] *М.Г. Бабенко, ORCID: 0000-0001-7066-0061 <mgbabenko@ncfu.ru>*
[1] *Е.И. Голимблевская, ORCID: 0000-0001-7188-8711 <elena.golimblevskaya@gmail.com>*
[1] *Е.М. Ширяев, ORCID: 0000-0002-2359-1291 <ea_or@list.ru>*
[1] *Северо-Кавказский федеральный университет,*
355017, Россия, г. Ставрополь, ул. Пушкина, д. 1
[2] *Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Повсеместное использование облачных технологий позволяет оптимизировать экономические издержки на содержание ИТ-инфраструктуры предприятий, но при этом увеличивает вероятность кражи конфиденциальных данных. Одним из механизмов для защиты данных от кражи

является криптография. Использование классических примитивов симметричного и ассиметричного шифрования не позволяет обрабатывать данные в зашифрованном виде. Для обработки конфиденциальных данных используют гомоморфное шифрование. Гомоморфное шифрование позволяет производить арифметические действия с зашифрованным текстом и получать зашифрованный результат, который соответствует результату операций, выполненных с открытым текстом. Одним из перспективных направлений для построения гомоморфных шифров является гомоморфные шифры, основанные на обучении с ошибками. В статье мы исследуем криптографические свойства существующих гомоморфных шифров (CKKS, BFV) на основе обучения с ошибками, сравниваем их технические характеристики: криптостойкость и избыточность данных, скорость кодирования и декодирования данных, скорость выполнения арифметических операций сложения и умножения данных, скорость выполнения операции KeySwitching.

Ключевые слова: гомоморфное шифрование; схемы полностью гомоморфного шифрования; система остаточных классов; BFV схема; CKKS схема; LattiGo; GoLang

Для цитирования: Бабенко М.Г., Голимблевская Е.И., Ширяев Е.М. Сравнительный анализ алгоритмов гомоморфного шифрования на основе обучения с ошибками. Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 37-52 (на английском языке). DOI: 10.15514/ISPRAS–2020–32(2)–4

Благодарности: Исследование выполнено при финансовой поддержке Российского научного фонда в рамках научного проекта № 19-71-10033.

1. Introduction

For large enterprises of any line of business, for smooth operation, it is necessary to maintain information channels, as well as timely processing and transmission of information. Currently, cloud technologies are often used to solve this problem. They allow storing information in one place giving access to several users simultaneously. To maintain the confidentiality of information (personal information of users, information that is a corporate secret, etc.) transmitted within the network, cloud services must provide a high level of cryptographic stability. However, the encryption and decryption of information takes a fairly high percentage of the computing power of the network, the increased load on which can have a negative impact on the operation of the entire network. The fact that homomorphic encryption allows performing calculations (changing) of information without having to decrypt it can increase performance and speed up the response time of the entire network, while maintaining a high degree of security.

Thus, the task is to find the most productive scheme of fully homomorphic encryption (FHE) [1-2]. Based on the analysis of the HE [3-8] schemes, we chose two for the research: Cheon, Kim, Kim and Song (CKKS) [12] and Brakerski/Fan-Vercauteren (BFV) [9], [10], [11]. Their main feature is the use of the Residue Number System (RNS) for performing operations in schemes. The study consists in measuring the execution time of the main functions in the schemes, thereby determining the most productive one. For the study, 4 sets of parameters are taken, which determine the dependence of performance on the degree of required security.

2. Background

For our research, schemes were selected that perform fully homomorphic encryption, and also have a Residue Number System (RNS) implementation [14].

2.1 Homomorphic encryption

Fully homomorphic encryption (FHE) is a form of encryption, which has to satisfy the additional requirement of homomorphism with respect to any operations on plaintexts. The encryption function $E(k, m)$, where m is a plaintext, k is an encryption key, is homomorphic with respect to the operation * on plaintexts, if and only if there is an effective algorithm M that receives any pair of ciphertexts of the form $E(k, m_1), E(k, m_2)$, which then produces a ciphertext c such that when decrypting c, plaintext $m_1 * m_2$ will be obtained [15]. A homomorphism with respect to the

operation + is defined similarly. When partially homomorphic cryptosystems are homomorphic with respect to only one plaintext operation, fully homomorphic systems support homomorphism with respect to both addition and multiplication [16]. This means that the following conditions are fulfilled for them:

$$\begin{cases} Dec\big(Enc(m_1) \otimes Enc(m_2)\big) = m_1 * m_2 \\ Dec\big(Enc(m_1) \oplus Enc(m_2)\big) = m_1 + m_2 \end{cases}$$

In this case, homomorphism in addition and multiplication operations is sufficient for the system to be fully homomorphic [16].

It is this feature of FHE that allows us to talk about the performance growth we are claiming, the ability to perform these operations on encrypted text can improve the performance of both cloud storage services and cloud computing centers.

2.2 Residue Number System

RNS is non-weighted number system based on modular arithmetic [17]. The representation of a number in RNS is based on a comparison of two integers modulo and the Chinese Remainder Theorem (CRT). Thus, RNS can be defined as a set of coprime moduli $(\rho_1, \rho_2, \dots, \rho_n)$, whose vector is called the base of RNS, as well as its product $P = \rho_1 \cdot \rho_2 \cdot \ldots \cdot \rho_n$, and so that every integer X that belongs to the range $[0, P-1]$ is associated with a set of residuals $(\alpha_1, \alpha, \dots, \alpha_n)$, where

$$\begin{aligned} \alpha_1 &\equiv X(mod\ \rho_1) \\ \alpha_2 &\equiv X(mod\ \rho_2) \\ &\dots\dots\dots\dots\dots\dots \\ \alpha_n &\equiv X(mod\ \rho_n) \end{aligned}$$

Also, based on the corollary of CRT, the uniqueness of the representation of non-negative integers from the interval $[0, P-1]$ is guaranteed [18].

In this case, the main advantage of representing numbers in the RNS is determined by the fact that such operations as addition, subtraction and multiplication can be performed by the formula:

$$A * B = (\alpha_1, \alpha_2, \dots, \alpha_n) * (\beta_1, \beta_2, \dots, \beta_n) =$$
$$= ((\alpha_1 * \beta_1)\ mod\ \rho_1), ((\alpha_2 * \beta_2)\ mod\ \rho_2), \dots, ((\alpha_n * \beta_n)\ mod\ \rho_n)$$

where * denotes operations such as addition, multiplication or subtraction [19]. These operations are called modular [19], since for them to be performed in RNS, one processing cycle of numerical values is sufficient. Moreover, this processing takes place in parallel, and the value of the number in each category is independent of other categories.

3. Homomorphic Encryption Schemes

To conduct research on HE schemes, we selected the LattiGo lattice-based cryptography library [13], written in the GoLang language. This library contains a set of functions that implement homomorphic encryption schemes. The LattiGo structure allows performing various studies on schemes, to experiment with both the complete schemes and individual operations performed in these schemes. All schemes meet publicly available safety standards. The parameters of it are presented below.

$N = 2^{\log N}$: the ring dimension. It determines cyclotomic polynomial degree. Moreover, it is the amount of polynomials coefficients in both plaintext and ciphertext. It can be only a power of two. N influences both security and performance. Because of this, for the schemes to work correctly, setting the parameter N requires special attention.

Q: ciphertext module. In LattiGo, it is selected as the product of small, relatively simple q_i moduli that provide $q_i \equiv 1\ mod\ 2N$, which makes representation in RNS and Number Theoretic Transform (NTT) possible. The choice of size of q_i modules is in the range from 50 to 60 bits,

which gives better performance. Q affects security and performance at the same time; if N is fixed and Q is bigger, this means reduced security and performance. Q is connected with N and must be carefully selected.

σ: variance, which is used for error polynomials. It influences security of the scheme.

It is also worth noting that all the schemes presented in this paper have a common HE base. Schemes perform arithmetic over the plaintext and ciphertext spaces. The plaintext space and the ciphertext space share a cyclotomic ring, which we denote in this paper as $\mathcal{H}$.

$$\mathcal{H} = \mathbb{Z}_Q[X]/(X^N + 1)$$

where N is a power of two.

Based on the arithmetic used by the scheme, various batch coding is performed. However, the mapping of arrays of numbers in a polynomial occurs with the property inherent in all HE schemes presented in this paper:

$$decode\big(encode(m_1) \otimes encode(m_2)\big) \approx m_1 \odot m_2$$

where $\otimes$ as a component-wise product, and $\odot$ as a nega-cyclic convolution.

3.1 CKKS Scheme

CKKS is a homomorphic encryption scheme used for approximate number arithmetic, suggested by Cheon, Kim, Kim, and Song. This scheme can be used for arithmetic over complex numbers and has RNS implementation. Batch encoding of this scheme is comparable to a cyclotomic ring as:

$$\mathbb{C}^{N/2} \leftrightarrow \mathcal{H}$$

Moreover CKKS depends on such parameter as plaintext scale. The values have to be scaled in order to be encoded on polynomial with integer coefficients. This parameter affects the accuracy of the output and the maximum permissible depth for the security parameter used. The configuration of parameters for CKKS is very dependent on the application, requiring a preliminary analysis of the scheme, performed in encrypted form, which in certain operations can reduce the scheme performance.

3.2 BFV Scheme

The Fan-Vercauteren version of scale-invariant Brakerski HE scheme is also considered in this paper. LattiGo contains RNS-implementation of this scheme. The scheme provides arithmetic over $\mathbb{Z}_t^N$. Then the batch encoding of this scheme, similar to the CKKS scheme, consists of the following:

$$\mathbb{Z}_t^N \leftrightarrow \mathcal{H}$$

If CKKS had only one independent parameter, then the BFV scheme has the following set of independent parameters:

P: expanded ciphertext module, which is used exclusively for the Mul operation (multiplication) and the like, with no effect on the degree of security. It is also defined as the product of small relatively simple moduli p_j and must be selected in such a way that $P > Q$ with a small margin (~ 20 bits), which is realized by means of one smaller Q module.

t: plaintext module. This module determines the value that is maximally possible for the plaintext coefficient. If the calculation leads to a higher value, then the value decreases modulo plaintext. For batching to work, the value must be simple and satisfy $t \equiv 1 \; mod \; 2N$. This module also does not affect security.

4. Analysis of HE Schemes Function performance

Benchmarking is performed for functions such as Encoding, Decoding, Encryption, Decryption, Addition, Multiplication, Relinearization and KeySwitching. Measurements are carried out for different dimensions of the encrypted vector (from 128 to 2048 numbers) and for different encryption parameters (table 1) [20].

Table 1: Security parameters

ID	$\log_2 N$	$\log_2 Q$	σ
1	12	109	3.2
2	13	218	3.2
3	14	438	3.2
4	15	881	3.2

4.1 Function Performance

4.1.1 Performance of Encoding Function

Measuring the performance of the Encoding function shows the superiority of the BFV scheme (table 2, fig. 1). The results were obtained in the experiment, due to which it can be noted that the performance of both schemes in this function does not significantly depend on the dimension of the encrypted vector. But as security options increase, the CKKS scheme requires more time for the Encoding function. This is due to the fact that in the CKKS scheme the encrypted vector consists of complex numbers, and encoding it in plaintext with integer coefficients requires more time than the similar procedure in the BFV scheme, which uses an integer vector.

Table 2. Encoding test results, ns

	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
CKKS	1	308963	333249	375870	460903	554734
	2	1873608	1973020	1884327	2002769	2215958
	3	6234737	6972733	6540090	7255575	7366968
	4	24415039	26627552	27666560	27404332	25905446
	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
BFV	1	211498	181524	169412	181010	171326
	2	496768	484148	511326	358980	472566
	3	1177199	1382577	959615	1301481	1067257
	4	3020121	3607336	3947700	2737189	3241723

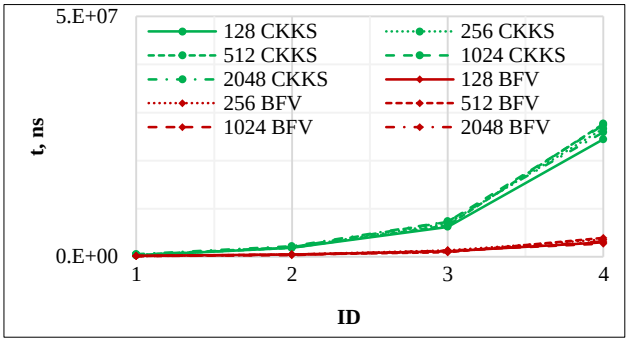

Fig. 1. Encoding time for BFV and CKKS

4.1.2 Performance of Decoding Function

The Decoding function depends on the dimension of the vector (table 3, fig. 2). The CKKS scheme scales worse due to the fact described above. The BFV scheme demonstrates the best speed of this function, but its advantage is negligible.

Table 3. Decoding test results, ns

	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
CKKS	1	1018856	1285869	1625036	2418919	4052705
	2	4200838	5038371	5152239	7399088	10315995
	3	13104508	14706263	14633906	17983167	21821681
	4	46158396	57014194	52938040	58693620	60117011
	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
BFV	1	1154013	1175882	1048750	1274209	1075534
	2	3151799	4236402	3288824	4564273	3038342
	3	12881134	16357558	14754335	13649037	12698237
	4	42709982	39141703	59220495	44049092	41901074

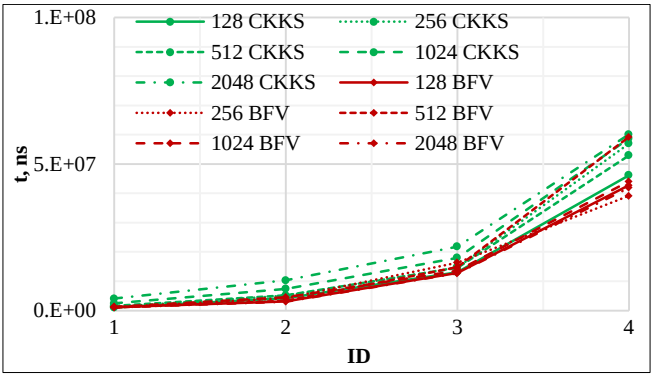

Fig. 2. Decoding time for BFV and CKKS

4.1.3 Performance of Encryption Function

Both encryption options (with public key and private key) have approximately the same performance on the first three sets of parameters. With the highest security requirement, in the case of secret key encryption, the BFV scheme scales worse and has lower performance (table 4). In the case of public key encryption, the results are reversed (table 5). In general, BFV is inferior in the performance of this function (fig. 3, 4). This situation is explained by the peculiarities of arithmetic implementation, which were described in subsection 3.2.

Table 4. Encryption with Secret Key test results, ns

	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
CKKS	1	2334489	2103892	2226278	2179083	2173156
	2	9766863	9382399	9218207	10142885	9507998
	3	33230264	33543617	31445147	32506263	33135191
	4	127892000	114750689	120734989	113363544	118239467
	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
BFV	1	2641997	3559443	2930780	3024797	3233360
	2	7644061	7330153	7071365	6598306	9201644
	3	34327793	34766696	37137252	36247603	36192216
	4	87575015	145189343	104613338	97278158	147942129

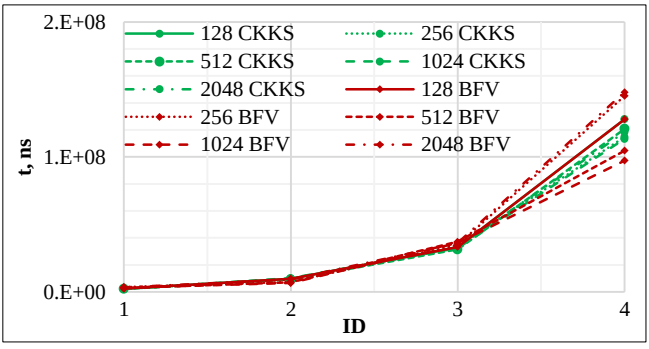

Fig. 3. Encryption Secret Key time for BFV and CKKS

Table 5. Encryption with Public Key test results, ns

	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
CKKS	1	2958304	2999553	2931131	3075510	3193087
	2	12985601	13081973	13171425	12353057	13008715
	3	46876542	52181155	42546888	44072000	43259396
	4	188649729	162992743	185340700	153663429	164277171
	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
BFV	1	3504304	3091037	2876235	3835154	4615228
	2	9948796	12968238	11532753	12323740	12444451

	3	45251935	45050544	42973388	49087468	34707847
	4	124654010	127437450	177076771	168856400	172971357

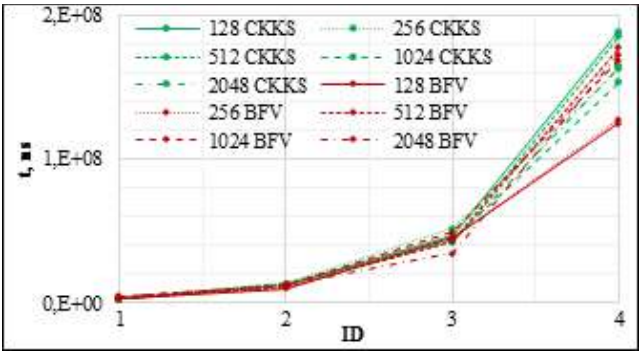

Fig. 4. Encryption Public Key time for BFV and CKKS

4.1.4 Performance of Decryption Function

When measuring the Decryption function, the results which are displayed in table 6 were obtained. Due to the fact that the BFV scheme uses NTT at the decryption stage, which the CKKS scheme uses in the Decode function, this function takes much less time with the CKKS scheme (fig. 5).

Table 6. Decryption test results, ns

	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
CKKS	1	75295	64490	91053	86014	83257
	2	704320	558029	500854	589965	569032
	3	1740359	1325015	1841163	1609382	1552320
	4	7188552	8656303	6451143	8596062	8852536
	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
BFV	1	1411973	1304798	1211739	1062796	893495
	2	4277947	3599959	3417547	3493461	3516575
	3	14734244	12412934	17919951	12627585	14222562
	4	79417746	81107353	59298720	81474821	82202793

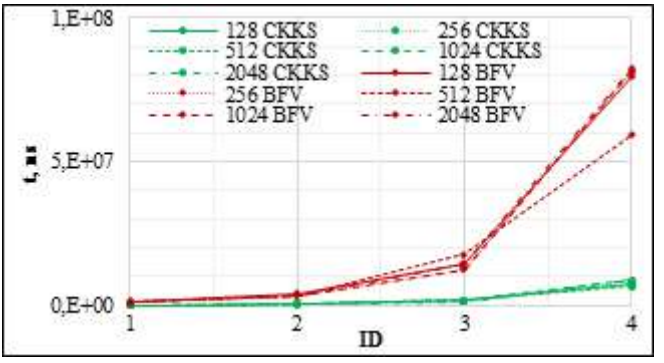

Fig. 5. Decryption time for BFV and CKKS

4.1.5 Performance of Addition Function

The study of the homomorphic addition function also shows the advantage of the CKKS scheme (table 7, fig. 6). This is because in the CKKS scheme data is properly scaled before the operation.

Table 7. Addition test results, ns

	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
CKKS	1	24465	37828	39856	30617	47407
	2	145437	235402	291096	312227	299579
	3	589837	981959	1081942	1020950	1071134
	4	2692325	3453714	3695143	2700388	4009002
	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
BFV	1	35960	34953	36248	35226	31980
	2	159519	113622	158053	123147	161107
	3	630680	635675	658730	442162	667723
	4	1724961	2314835	1881167	2609253	2738442

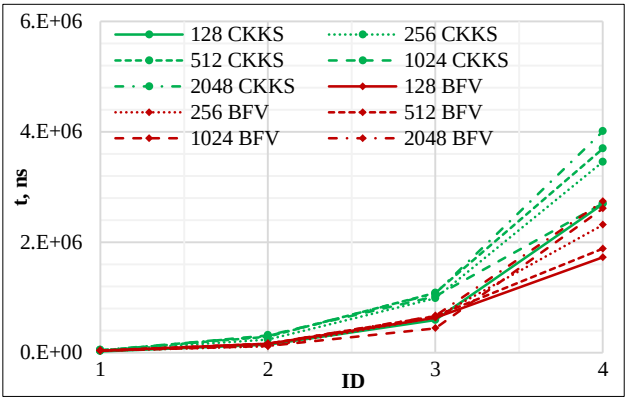

Fig. 6. Addition time for BFV and CKKS

4.1.6 Performance of Multiplication Function

Simulation of the multiplication of ciphertext by a scalar and by ciphertext showed two different results (table 8, 9, fig. 7, 8)). When multiplied by a scalar, BFV scheme shows high performance. But the BFV scheme has scalar multiplication only for the uint64 type, while CKKS provides a solution for types like complex128, float64, uint64, int64 and int. The constant is represented as a complex number, when multiplying the ciphertext by which, obviously, it takes more time than when multiplying by a constant of the uint64 type in the BFV scheme.

Table 8. Multiplication by Scalar test results, ns

	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
CKKS	1	42654	62583	70964	67642	63079
	2	231795	503098	545495	518164	550229
	3	1069628	1745948	1756096	1705708	1733524
	4	4077041	6304774	6041801	6270333	6130658
	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
BFV	1	48430	48057	43500	48024	45157

	2	219919	170531	217853	186792	213763
	3	854740	815247	649023	879803	839350
	4	2831192	3393249	2587298	2762533	2592716

At the same time, the multiplication of ciphertext by ciphertext in the CKKS scheme has higher performance, and also, compared to BFV, a relatively small increase in time spent with increasing security settings.

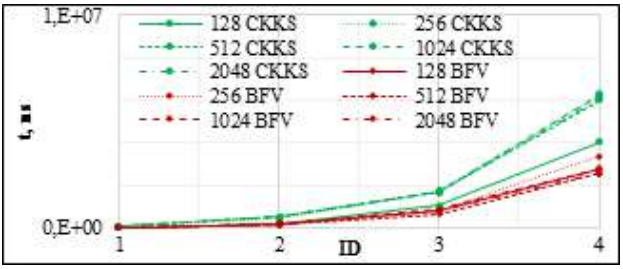

Fig. 7. Multiplication by Scalar time for BFV and CKKS

Fig. 8. Multiplication time for BFV and CKKS

Table 9. Multiplication test results, ns

	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
CKKS	1	134293	216760	180038	171766	205594
	2	674873	1469598	1525148	1078809	1152720
	3	3782338	4913585	5253617	5065445	3949636
	4	13078065	12682334	17907985	15632646	17966423
	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
BFV	1	7089414	7487352	7695449	10207294	7628154
	2	31767778	25332290	35383221	37529547	33403094
	3	170244000	170229078	158613588	171963733	180915567
	4	932056250	967022400	925598350	556013800	951451000

4.1.7 Performance of Relinearization Function

From fig. 9 and table 10 we can see that the Relinearization function in the BFV scheme with higher safety parameters is more productive. But on the first set of parameters, the implementation of the CKKS scheme wins.

Table 10. Relinearization test results, ns

	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
CKKS	1	1687438	2661698	3424449	2261363	3231668
	2	18201540	34499628	30833154	30375355	30167662
	3	100644400	154891650	153884971	103854875	146318000
	4	383689733	442374400	633888950	658028000	666219500
	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
BFV	1	3724441	2389712	3317893	4066252	3757963
	2	11464962	11950768	11067133	12411460	12308602
	3	69749413	70541320	75965527	61616068	75667007
	4	290493225	292239375	350729100	364713920	306935225

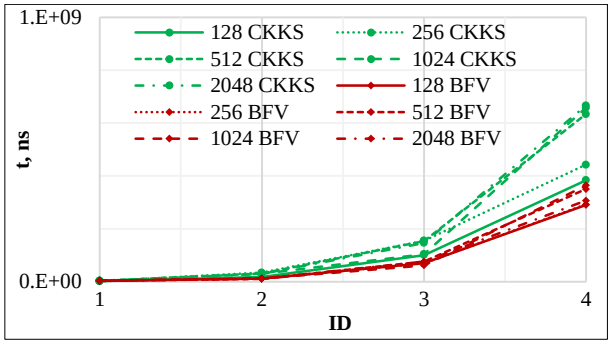

Fig. 9. Relinearization time for BFV and CKKS

4.1.8 Performance of KeySwitching Function

In this study, the CKKS scheme showed higher performance (table 11. Fig. 10). This function is faster with complex numbers, not integers, since the complex number field is easier to scale.

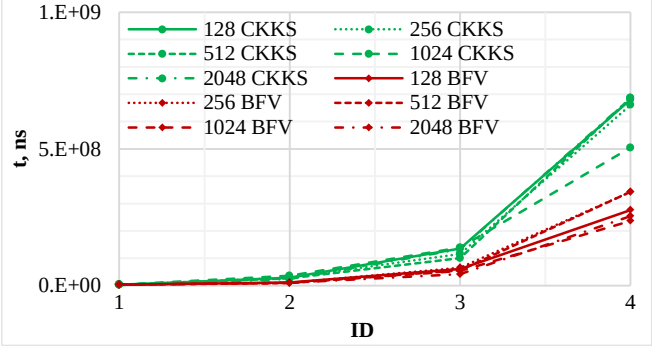

Fig. 10. KeySwitching time for BFV and CKKS

Table 11. KeySwitching test results, ns

	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
CKKS	1	2503032	3026201	3224305	3203122	2546295

	2	28739441	28862725	26122406	35456117	23952320
	3	135554285	115695133	100226020	138455820	133445758
	4	679806800	660336400	686261350	504191200	685178233
BFV	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
	1	2938459	3010961	3227951	2688819	3374234
	2	10792889	9802067	8450965	10495674	10323025
	3	60385157	65758147	55269293	53448786	41285438
	4	276312900	342915800	342230575	236232667	254161425

4.2 Signal to Noise Ratio (SNR)

It is known that the ciphertext obtained with HE has redundancy. Thus, a degree of this redundancy is an important parameter of the HE schemes. This redundancy can be determined by examining SNR.

4.2.1 SNR of Plaintext to Vector

When conducting SNR of the plaintext to the initial vector, a high degree of redundancy was found in both schemes, but the CKKS scheme showed a better ratio (table 12, fig. 11).

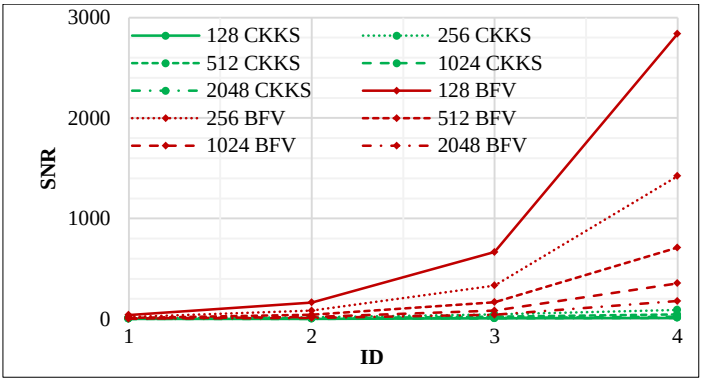

Fig. 11. SNR of Plaintext to Vector

Table 12. SNR of Plaintext to Vector

CKKS	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
	1	1.049194	8.38208	4.185791	2.090729	1.04755
	2	2.093929	16.74509	8.37439	4.188866	2.092964
	3	5.538696	44.25195	22.10657	11.06628	5.530708
	4	11.06649	88.49725	44.25009	22.12093	11.06154
BFV	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
	1	39.62492	19.81713	9.907773	4.958288	2.474978
	2	164.451	82.67813	41.0744	20.5674	10.2885
	3	665.6461	333.0266	166.6605	83.25319	41.63363
	4	2838.382	1422.057	710.3599	354.8242	177.552

4.2.2 SNR of Ciphertext to Plaintext

Within plaintext encryption, the noise in the CKKS scheme increases much more than in the BFV scheme. But with higher security settings, its increase is reduced.

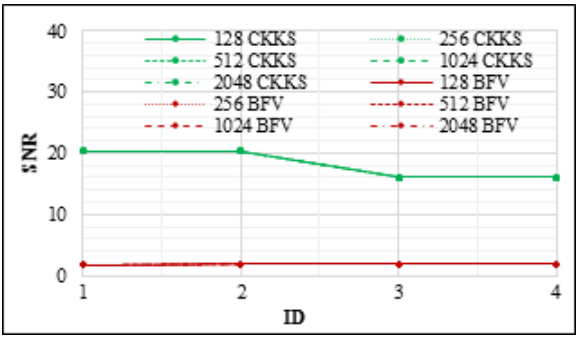

Fig. 12. SNR of Ciphertext to Plaintext

Table 13. SNR of Ciphertext to Plaintext

	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
CKKS	1	20.30157	20.34031	20.34885	20.37499	20.34864
	2	20.35725	20.36439	20.36092	20.35067	20.36643
	3	16.05651	16.08642	16.1052	16.08278	16.08872
	4	16.08454	16.09035	16.09035	16.09291	16.09277
	ID	Dimension of encrypted data vector				
		128	256	512	1024	2048
BFV	1	1.9481	1.949288	1.949583	1.948348	1.948181
	2	1.963438	1.963153	1.962645	1.963236	1.962972
	3	1.963446	1.96309	1.963343	1.963505	1.963333
	4	1.965543	1.965856	1.965686	1.965566	1.965581

In general, it can be noted that with lower security settings and a larger dimension of the ciphertext, the BFV scheme shows less data redundancy. But in the case of high security settings, the CKKS scheme is clearly more profitable.

5. Conclusion

The use of cloud technology on the one hand can reduce the cost of maintaining IT infrastructure, but on the other hand it has a number of limitations associated with the scope of application. For example, when processing confidential data, it is necessary to consider the risks of information theft, to reduce the probability of data theft, homomorphic encryption is used. In our work we investigate two schemes of CKKS and BFV homomorphic data encryption from the point of view of technical characteristics, such as data encoding speed, data decoding speed, encryption and decryption speed, speed of arithmetic operations with encrypted texts, evaluates noise parameters that occur when encrypting data. A comparison of the two schemes shows that there is no one-size-fits-all approach that can be used as a universal solution. Further we plan to investigate a question of realization of matrix operations with use of various homomorphic encryption schemes.

References / Список литературы

[1] Craig Gentry. A Fully Homomorphic Encryption Scheme. PhD thesis, Stanford University, 2009, 199 p.

[2] Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. Lecture Notes in Computer Science, vol. 9814, 2016, pp. 153-178.

[3] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully Homomorphic Encryption over the Integers. In Proc. of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2010, pp. 24-43.

[4] Zvika Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. Lecture Notes in Computer Science, vol. 7417, 2012, pp. 868-886.

[5] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. In Proc. of the forty-fourth Annual ACM Symposium on Theory of Computing, 2012, pp. 1219-1234.

[6] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. IACR Cryptology ePrint Archive, 2012:144, 2012.

[7] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from Learning with Errors: Conceptually-Simpler, Asymptotically Faster, Attribute-Based. Lecture Notes in Computer Science, vol. 8042, 2013, pages 75-92. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption without Bootstrapping. ACM Transactions on Computation Theory (TOCT), vol. 6, issue 3, 2014, article no. 13.

[8] Tancrède Lepoint, Michael Naehrig. A Comparison of the Homomorphic Encryption Schemes FV and YASHE. Lecture Notes in Computer Science, vol. 8469, 2014, pp. 318=335.

[9] Jean-Claude Bajard, Julien Eynard, M Anwar Hasan, and Vincent Zucca. A Full RNS Variant of FV Like Somewhat Homomorphic Encryption Schemes. Lecture Notes in Computer Science, vol. 10532, 2016, pp. 423-442.

[10] Shai Halevi, Yuriy Polyakov, and Victor Shoup. An Improved RNS Variant of the BFV Homomorphic Encryption Scheme. Lecture Notes in Computer Science, vol. 11405, 2019, pp. 83-105.

[11] Hao Chen, Kim Laine and Rachel Player. Simple Encrypted Arithmetic Library-SEAL (v2.1). Lecture Notes in Computer Science, vol. 10323, 2017, pp. 3-18.

[12] Lattigo: lattice-based cryptographic library in Go. Available at: https://github.com/ldsec/lattigo, accessed: 10.05.2020.

[13] Behrooz Parhami. Computer Arithmetic: Algorithms and Hardware Design. 2nd edition, Oxford University Press, New York, 2010. 641+xxv p.

[14] Varnovsky N.P., Shokurov A.V. Homomorphic Encryption. Trudy ISP RAN/Proc. ISP RAS, vol. 12, 2007, pp. 27-36 (in Russian) / Варновский Н.П., Шокуров А.В. Гомоморфное шифрование. Труды ИСП РАН, том 12, 2007 г., стр. 27–36.

[15] Babenko L.K., Burtyka Ph.B., Makarevich O.B., Trepacheva A.V. Fully Homomorphic Encryption Techniques using Matrix Polynomials. Voprosy kiberbezopasnosti, no. 1(9), 2015, pp. 14-25 (in Russian) / Бабенко Л.К., Буртыка Ф.Б., Макаревич О.Б., Трепачева А.В. Методы полностью гомоморфного шифрования на основе матричных полиномов. Вопросы кибербезопасности, no. 1(9), 2015 г., стр. 14-25.

[16] Erdnieva N.S. The use of special modules of the system of residual classes for redundant representations. Vestnik of Astrakhan State Technical University. Series: Management, Computer Science and Informatics, № 2, 2013, pp. 75-85 (in Russian) / Эрдниева Н.С. Использование специальных модулей системы остаточных классов для избыточного представления. Вестник Астраханского государственного технического университета. Серия: управление, вычислительная техника и информатика, № 2, 2013 г., стр. 75-85.

[17] Sagalovich Yu. L. Introduction to algebraic codes, 2nd ed. M., IPPI RAS, 2011, 302 p. (in Russian) / Сагалович Ю. Л. Введение *в* алгебраические коды, 2-е изд. М., ИППИ РАН, 2011, 302 с.

[18] Lavrinenko A.N., Chervyakov N.I. Research of non-modular operations in the system of residual classes. Belgorod State University Scientific Bulletin. Series: Economics. Computer science, no. 1(120), 2012, pp. 110-122 (in Russian) / Лавриненко А.Н., Червяков Н.И. Исследование немодульных операций в системе остаточных классов. Научные ведомости Белгородского государственного университета. Серия: Экономика. Информатика, no. 1 (120), 2012 г., стр. 110-122.

[19] Martin Albrecht et al. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, 2018, 33 p.

Information about authors / Информация об авторах

Mikhail Grigorievitch BABENKO graduated from Stavropol State University in 2007. He defended his thesis in 2011. Currently he is a lecturer of the Department of Applied Mathematics and Mathematical Modeling of the North Caucasus Federal University. Research interests: Algebraic structures in the Galois fields, modular arithmetic, neurocomputer technologies, digital signal processing, cryptographic methods for protecting information.

Михаил Григорьевич БАБЕНКО окончил Ставропольский государственный университет в 2007 году. Защитил кандидатскую диссертацию в 2011 г. Преподаватель кафедры прикладной математики и математического моделирования Северо-Кавказского федерального университета. Сфера научных интересов: алгебраические структуры в полях Галуа, модулярная арифметика, нейрокомпьютерные технологии, цифровая обработка сигналов, криптографические методы защиты информации.

Elena Igorevna GOLIMBLEVSKAIA is a student of the North Caucasus Federal University of the Department of Information Systems and Technologies since 2017. Research interests: modular arithmetic, neurocomputer technologies, cryptographic methods of information protection.

Елена Игоревна ГОЛИМБЛЕВСКАЯ в 2017 году поступилd в Северо-Кавказский федеральный университет на кафедру информационных систем и технологий. Сфера научных интересов: модулярная арифметика, нейрокомпьютерные технологии, криптографические методы защиты информации.

Egor Mikhailovitch SHIRIAEV is a student of the North Caucasus Federal University of the Department of Infocommunication since 2016. Research interests: modular arithmetic, neurocomputer technologies, cryptographic methods of information protection.

Егор Михайлович ШИРЯЕВ в 2016 году поступил в Северо-Кавказский федеральный университет на кафедру инфокоммуникаций. Сфера научных интересов: модулярная арифметика, нейрокомпьютерные технологии, криптографические методы защиты информации.

DOI: 10.15514/ISPRAS-2020-32(2)-5

Test environment for verification of multi-processor interrupt system with virtualization support

D.A. Lebedev, ORCID: 0000-0002-9244-4949 <lebedev_d@mcst.ru>
V.N. Kutsevol, ORCID: 0000-0001-7322-2622 <kutsevol_v@mcst.ru>
MCST, 1, Nagatinskaya st., Moscow, 117105, Russia

Abstract. Interrupt system is an important part of microprocessors. Interrupts are widely used for interaction with hardware and responding to stimuli. Modern microprocessor interrupt systems include hardware support of virtualization. Hardware support helps to increase the performance of virtual machines. However, including additional functionality may lead to potential errors. The paper presents an overview of approaches used for multi-core microprocessors interrupt system with virtualization support verification. Some definitions and characteristics of interrupt systems that needed to be taken into account in the process of verification are described. Stand-alone verification environment general scheme is presented. Universal Verification Methodology was applied to construct test system. To simplify development of checking module discrete-event with time accounting reference model was used. Sequences of primary requests and automatically generated secondary requests in the special modules named auto-handlers were used for test system behavior randomization. We describe some difficulties discovered in the verification process and corresponding solving methods. Generalized test algorithm stages are presented. Some other techniques for checking the correctness of interrupt system have been reviewed. In conclusion, we provide the case study of applying the suggested approaches for interrupt system verification of microprocessors with "Elbrus" and "SPARC-V9" architectures developed by MCST. The results and further plan of the test system development are presented.

Keywords: test environment; standalone verification; multicore microprocessors; interrupt system; UVM; virtualization

For citation: Lebedev D., Kutsevol V. Test environment for verification of multi-processor interrupt system with virtualization support. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 53-60. DOI: 10.15514/ISPRAS–2020–32(2)–5

Тестовое окружение для верификации многопроцессорной системы прерываний с поддержкой виртуализации

Д.А. Лебедев, ORCID: 0000-0002-9244-4949 <lebedev_d@mcst.ru>
В.Н. Куцевол, ORCID: 0000-0001-7322-2622 <kutsevol_v@mcst.ru>
АО «МЦСТ», 117105, Москва, Россия, ул. Нагатинская, д. 1

Аннотация. Система прерываний является важной частью микропроцессоров. Прерывания широко используются для взаимодействия с оборудованием и реагирования на сигналы. Современные микропроцессорные системы прерываний включают аппаратную поддержку виртуализации. Аппаратная поддержка помогает повысить производительность виртуальных машин. Однако добавление дополнительной функциональности может привести к появлению потенциальных ошибок. В статье представлен обзор подходов, используемых для систем прерывания в многоядерных микропроцессорах с аппаратной поддержкой виртуализации. Описаны некоторые определения и характеристики систем прерываний, которые необходимо учитывать в процессе проверки. Представлена общая схема автономной среды верификации. Universal Verification Methodology была применена для построения тестовой системы. Для упрощения разработки модуля проверки

использовалась эталонная модель с учетом временных характеристик. Последовательности первичных запросов и автоматически генерируемые вторичные запросы в специальных модулях авто генерации использовались для рандомизации поведения тестовой системы. Были описаны некоторые трудности, обнаруженные в процессе верификации, а также соответствующие методы их решения. Представлены обобщенные этапы алгоритма тестирования. Были рассмотрены некоторые другие методы проверки корректности работы системы прерываний. В заключение приведены примеры применения предложенных подходов для верификации системы прерываний микропроцессоров с архитектурой Эльбрус и «SPARC-V9», разработанной АО МЦСТ. Представлены результаты и дальнейший план развития тестовой системы.

Ключевые слова: тестовая система; автономная верификация; многоядерные микропроцессоры; система прерываний; UVM; виртуализация

Для цитирования: Лебедев Д.А., Куцевол В.Н. Тестовое окружение для верификации многопроцессорной системы прерываний с поддержкой виртуализации. Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 53-60 (на английском языке). DOI: 10.15514/ISPRAS–2020–32(2)–5

1. Introduction

State of the art microprocessors are becoming complex systems. The development of new technological processes allows integrating great number of controllers and subsystems on the one crystal. The logic of their work becomes more complicated. As an example: interrupts delivery and handling mechanisms. Interrupts are widely used for interaction with hardware and responding to stimuli. Interrupt system is an important part of microprocessor and have to be tested thoroughly because errors in this block may lead to a race condition or multiple accesses to shared memory [1].

Virtualization is necessary for modern tasks like cloud computing, modeling, information security, and other scientific researches. Interrupt system hardware support for virtualization is implemented in the most of modern SoC (System on a Chip) [2 3]. Performance requirements for virtualized systems are steadily increasing [4]. One method to increase performance of virtual operating system is to implement hardware support. However, this becomes an additional error-prone place in the microprocessor system. In addition, I/O virtualization is a difficult part of system virtualization due to the increasing number of I/O devices attached to the computer and the increasing diversity of I/O device types [5]. A significant part of the total number of interrupts is accounted for I/O interrupts.

Many approaches are proposed for verification of the interrupt system. In [6] authors divide existing techniques for verifying interrupt systems into two categories. First of them is testing via executing some programs and invocation various interrupts. The disadvantage of this approach is probability of missing important bugs. The second one is constructing and analyzing formal models. As was already mentioned in [7] constructing and analyzing of formal models are complicated and requires detailed specification. In this paper, we propose simulation-based methods for verification of interrupt system that are much simpler and could be applied at earlier stage of RTL (Register Transfer Level) model development when first versions of specification were available.

In [8] verification of the interrupt system provided using interrupt-driven software which launched on the whole microprocessor system. This method gives good results in finding race condition bugs but requires fine-tuning of interrupt sending schedule. Moreover, none of the discussed methods recreates rare dynamic scenarios. Stand-alone verification usually used for building necessary test conditions to achieve sufficient testing quality.

There are a number of methods to implement a standalone functional verification [7]. In this paper, we focus on building testing environment for interrupt system using Universal Verification Methodology (UVM) [9]. UVM is IEEE standard elaborated by Accellera Systems. UVM is a set of class libraries defined using the syntax and semantics of SystemVerilog hardware description

and verification language. The verification environment built using UVM divided into specific components each of them performs its own role in test scenario. One of the main advantages of UVM-build verification environments is reusability of components. It helps to support probable changes in DUV (Device Under Verification). The main drawback of UVM is a complexity of its learning. Our verification team have a number of already debugged components and classes. Therefore, we can apply UVM for developing interrupt system stand-alone verification environment.

The rest of the paper is organized as follows. Section 2 reviews some definitions and describes general technique for standalone verification of the interrupt subsystem. Section 3 describes a case study and suggests approaches for functional verification of interrupt system. Section 4 describes additional used approaches. Section 5 reveals results and Section 6 concludes the paper.

2. Definitions and verification methods

Let us give a few definitions. An i*nterrupt* is asynchronous signal that indicates necessity for control transferring to some external to the processor core requester. An *interrupt vector* characterizes the transmitted signal. Interrupt vector points to the memory area where the corresponding *interrupt handler* is located. The interrupt handler is a code that should be executed instead of current main program. It essential to mention that interrupts change main memory and device registers state but main context of processor work stays the same.

We can divide interrupts into two types: *non-maskable* and *maskable.* Non-maskable interrupt cannot be disabled. It has more priority then maskable and used for exceptional conditions like critical faults, system handling and other. Maskable interrupt is intended for maintenance of system and user programs: the organization of external exchanges, interaction of different processes and working with timers. Maskable interrupts usually have several levels of priority. If an interrupt signal is received, but its priority is less than the one that came earlier, the interrupt becomes *pending*. Important time characteristic is *interrupt latency*. This is a time interval from the start of the interrupt request to the start of the interrupt handler execution. A set of system setups such as interrupt enabling, current priority, and the presence of a large number of pending interrupts can cause an *interrupt loss* or *spurious interrupt*. The spurious interrupt is an invalid, short-duration signal on an interrupt input. It is necessary to take into account these features when verifying interrupt system.

It is necessary to define some concepts related to virtualization. *Hypervisor* is a system software that distributes physical resources between *virtual machines*. A computer on which a hypervisor handles one or more virtual machines is called a *host*, and each virtual machine is called a *guest*. As a part of hardware interrupts support, the guest can be bounded on one or more cores. These cores are called *guest cores*. Without hardware support, the virtual software model of the interrupt controller, register requests, and interrupt delivery involved interception and emulation. Because of this, the performance of the VM is reduced.

We isolate a part of microprocessor system while providing stand-alone verification. The device specification have to describe correct sequence of stimuli and reactions in different device states. All interactions controlled by test environment – a program that generates input stimuli, checks correctness of reactions and calculates the quality of testing. Therefore, test environment could be divided into separate modules each performing its own function:

- input stimuli generator;
- correctness checking module;
- coverage collector.

Usually the interrupt controllers are handled by a set of software-visible registers. All requests to registers processed sequentially strictly one at a time. Thus, input generator is responsible not only for sending primary requests it also sets up a device. Next, we need to collect device responses and process them correctly. Generation of stimulus and collection of reactions simplified by using

Transaction Level Modeling (TLM) [10]. This method allows concentrating on the interaction functionality with DUV. It is necessary to implement only once how to handle with interface and then simple data sending and receiving functions are used. Collected information about functional code coverage is used to identify untapped regions of controller during testing. Analyzing that information helps to refine test scenarios and add new ones. This approach is called coverage driven constrained random verification [11].

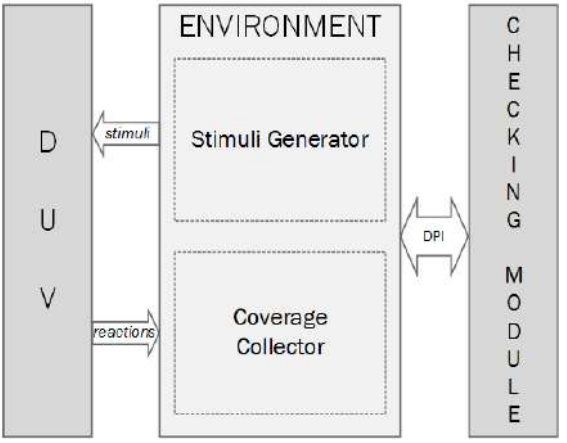

Fig 1. Generalized scheme of test environment

Checking of behavior correctness is a complex and an important part while providing interrupt system stand-alone verification. It is usually easier to build test environment external to the reference model for this purpose. Identical input stimuli fed into the DUV and reference model. If reactions differ, it indicates a possible error in the system. Reference models usually written in high-level language (C, C++) or some specialized languages for hardware verification, such as SystemVerilog, SystemC or «e». The reference models could be divided into three types: *cycle-accurate*, *discrete-event with time accounting* and *event models* [12]. The type of verified device defines the type of the reference model. Cycle-accurate and discrete-event with time accounting models require specification with describing behavior on a register transfer level. Development of models of these types is labor-intensive when the design specification is changing. However, because the interrupt system includes the use of counters, we have to choose more accurate reference model. To simplify development of checking module we use discrete-event with time accounting reference model. Communication between reference model and test environment carried out with DPI (Directed Programming Interface). The use of DPI is necessary to coordinate the variable types used in different programming languages. Exchange of test data occurs instantly by calling appropriate functions. Generalized scheme of test environment shown on fig. 1.

3. Using gray box approach for verification of home memory unit

Elbrus Programmable Interrupt Controller (EPIC) is a device intended for capturing, storing external and interprocessor interrupts and delivering them to the microprocessor cores. EPIC is a part of 16-core microprocessor with “Elbrus” architecture developing by MCST. Software-available registers manage the controller setup and handling of interrupts. The algorithms for working with maskable and non-maskable interrupts are different. Maskable interrupts have four priority classes. An unhandled interrupt with the highest priority are placed on CIR (Current Interrupt Register) or if CIR is busy placed on PMIRR (Pending Maskable Interrupt Request Registers). Signal to the core sets only if current core priority is less than interrupt priority in CIR register. Interrupt system implements hardware support of virtualization. Supporting options include additional control registers representing the state of guest interrupt system and guest-physical core mapping table.

As we mentioned before, test system for stand-alone verification of the interrupt system is based on UVM. UVM helps to setup system, generate pseudo-random constrained input requests and monitor all changes of device states. Input stimuli generation is usually implemented at level that is more abstract than register transfers and interface signals. Input and output device ports combined into interfaces based on the similarity of the performed functions. Transaction level packets are transferred on interfaces [13]. Serialization and deserialization modules transform transaction level packets to signal level interfaces.

In modern microprocessor system, there are several sources of interrupt requests or primary requests. Each interrupt source is replaced by a special test sequence. Generation of primary requests should be similar to that in real microprocessor system. One of the benefits of stand-alone verification is ability to create highly loaded test scenarios relatively easily. The efficiency of verification increases because generated pseudo-random requests can cover most of the edge cases. We use sequences of primary requests and generate secondary requests automatically in the special modules named *auto-handlers*. In case of interrupts system secondary request is a writes and reads sequence to the registers that simulating a real software behavior. Interrupts generated from primary requests are monitored and collected in special buffers. After that auto-handler randomly choose next interrupt to handle. Thus, an interrupt latency parameter randomly changes.

For virtualization support, the hardware implements a DAT (Destination Address Table) that stores the number of the currently active guest core for each physical core. Hypervisor manages the table by requests to the registers. The state of the table has to be the same in all processors of multiprocessor system. This is implemented through hardware-generated control messages. Auto-handler module with randomized parameters of sending service messages were added to verify the described above functionality.

Maskable and non-maskable interrupt handling is differ. For maskable interrupts, it is necessary to restore the previous core priority after handling current interrupt. For this purpose, we add monitoring module with memory where ratio “number of core-current priority” stored.

There was a dynamic inconsistency problem between the RTL implementation of the controller behavior and the reference model. Under dynamic test conditions, there may be situations when we start to handle a maskable interrupt *m.handle(x)*, read vector value from RTL-model register and it is not equal to reference model value. For this situation, we introduce additional function *m.correct(x)* that transfer RTL-model value to the reference model. During a test, the reference model accumulates possible vectors. When there is a values inconsistency situation the special algorithm in reference model iterates over possible vector values and makes a decision about correction. The pseudo-code of the algorithm is presented below.

```
Vector correctness check:
while true do
   wait m ← start(x)
   m.handle(x)
   if check(x !=x`) then
      m.correct(x)
   else m.print_ok(x)
end
```

Register for non-maskable interrupts contains several types of interrupts and they represented as one bit for an interrupt. In high-load dynamic tests with many non-maskable input requests for interrupts there may be differences in a register content. A similar procedure for correction *nm.correct(x)* is performed for non-maskable interrupts. From a functional point of view, handling guest interrupts does not differ from the procedures described above. The only difference is setting the bit that indicates a guest when working with registers.

Another problem is an interrupt vectors overlay. This is a subtype of dynamic inconsistency problem when the same vector values are imposed on each other. This can happen for example when working with cyclic timers. Additional functionality was added to the RTL and reference

models of verified device. A special module and interface signaling about interrupt overlay monitored in the test environment and then transferred to the reference model. Method when we use hints from a verified device for single correct state identification named "gray box" method [7, 11]. The information about overlay type used by reference model to exclude extra interrupts. This method required direct involvement of the device developer and a detailed description of the interface.

In a multiprocessor system, the interrupt system settings in each processor may differ. It is possible to configure the presence and numbers of processors and cores. It is necessary to check all possible system setups for completeness of verification. Interrupt routing changes when we change the processor or core numbering. This adds restrictions on generating primary requests. To simplify primary requests generation, we added a set of functions that automatically capture changes in system settings and allow easily select possible interrupt directions.

The generalized test algorithm is presented below:

1. randomization of device configuration;
2. configuring registers;
3. configuring guest cores;
4. choosing random requester and presence of receiver;
5. choosing random type of interrupt;
6. sending primary requests for interrupt, collecting reactions, starting auto handling;
7. transferring transaction information to reference model on each step of algorithm.

*4. Additional verification methods*Special cases

To check the interrupt system functionality related to virtualization support it was necessary to create special tests scenarios. For example, hypervisor can remove a guest from core if it receives an intercept signal. In this moment guest may contain unfinished or unhandled interrupts. In real system after bounding the guest back, we needed to recover it previous state. Special test cases of this kind play a big role in verifying correctness of controller operating. The development of algorithms for such narrowly focused tests is possible due to the availability of well-described documentation, analysis of the functional coverage of the code, and discussions of the strategy with the device developer.

4.2 Assertions

SystemVerilog Assertions (SVA) is a part of SystemVerilog [14]. The assertions are used to specify the behavior of the test environment and DUV interfaces. Parts of a verification environment have to implement certain functions. We can add assertions to check correctness of the module. Violation of an assertion signals about an error. Usage of assertions is an effective method of error detection especially in the beginning of the project. In addition, assertions alert about uncertain and unconnected states of interface signals.

4.3 After test checking

Communication between test environment and reference model is provided using DPI. Special buffers and memories contain generated answers form different interfaces. The correct behavior of the DUV and reference model determined in providing certain number of responses. After test scenario ending we check an absence of transactions in these communication buffers. Detection of extra number of requests signals about a potential error either in the verified device or in the reference model.

5. Results

The approaches described in this paper were applied for standalone verification of interrupt system of the 16-core microprocessor with “Elbrus” architecture and 2-core microprocessor with “SPARC-V9” architecture.

There is some difference in interrupt systems in these microprocessors. The 16-core microprocessor’s interrupt system has a hardware support of virtualization and specialized interfaces for handling requests from virtual OS and hypervisor.

The 2-core microprocessor does not support virtualization. Most of interfaces differs from the “Elbrus” interrupt system. One of its features is an additional module that handles direct MSI-X interrupts and a separate register interface for handling MSI-X interrupts. The test environment based on UVM made it relatively easy to change the format of modules that work with the interfaces while preserving their functionality.

“SPARC-V9” implementation of the interrupt system contains an additional synchro signal. The parts of the interrupt system in which several synchro signals interact should be checked carefully. At the beginning of each test, we generate random periods of synchro signals and their shifts that are relative to each other. Ranges of each synchro signal have to be described in the device specification. This method helped us to detect synchronization errors in RTL-model internal modules.

In the process of the standalone verification of the interrupt systems, we verified not only RTL-models. Parts of reference models were used in full-system “Elbrus” and “SPARC” machine simulators. Aforementioned approaches helped to find and correct some errors in the simulators. Interrupt systems distribution of errors is presented in Table 1. Code functional coverage was carried out and for “Elbrus” it was 94%, for SPARC 96% coverage was extracted.

Table 1. Distribution of errors and its quantity

Verified object	Number of bugs
RTL Elbrus	84
Elbrus Simulator	163
RTL SPARC	24
SPARC Simulator	23

6. Conclusion and directions for future work

Interrupt system with hardware virtualization support is one of important parts of modern microprocessors. The correct operation of the interrupt system allows avoiding critical errors and improves performance of virtual machines.

In this paper, we have presented a stand-alone test environment for interrupt system based on UVM. The proposed approaches could be used to verify interrupt systems of different multicore microprocessors regardless of their architectures. Developed test environment and test scenarios made it possible to detect and correct a number of errors that were not detected by other verification methods.

In the future, we plan to enhance error diagnostics and adapt the test environment for the forthcoming projects.

References / Список литературы

[1] Makoto Higashi, Tetsuo Yamamoto, Yasuhiro Hayase, Takashi Ishio, and Katsuro Inoue. An effective method to control interrupt handler for data race detection. In Proce. of the 5th Workshop on Automation of Software Test, 2010, pp. 79–86.

[2] ARM Generic Interrupt Controller Architecture Specification version 4.0, 2107, available at: https://static.docs.arm.com/ihi0069/c/IHI0069C_gic_architecture_specification.pdf, accessed 25.05.2020.
[3] Intel Virtualization Technology for Directed I/O, Architecture Specification. Intel, 2019, available at: https://software.intel.com/sites/default/files/managed/c5/15/vt-directed-io-spec.pdf, accessed 25.05.2020.
[4] Znamenskiy D.V. Alternatives of hardware virtualization support inplementation for Elbrus processor architecture. Voprosy radioelektroniki, 2014, vol. 4, no. 3, pp. 64‑73 (In Russian) / Знаменский Д.В. Выбор вариантов реализации средств аппаратной поддержки виртуализации архитектуры «Эльбрус». Вопросы радиоэлектроники, том 4, no. 3, стр. 64–73.
[5] Hennessy J.L., Patterson D.A. Computer Architecture: A Quantitative Approach. Fifth Edition. Morgan Kaufmann, 2012. 857 p.
[6] Chungha Sung, Markus Kusano, Chao Wang. Modular Verification of Interrupt-Driven Software. In Proc. of the 32nd IEEE/ACM International Conference on Automated Software Engineering, 2017, pp. 206–216.
[7] Lebedev D.A., Petrochenkov M.V. Test environment for verification of multi-processor memory subsystem unit. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 3, 2019. pp. 67-76. DOI: 10.15514/ISPRAS-2019-31(3)-6.
[8] John Regehr. Random testing of interrupt-driven software. In Proc. Of the International Conference on Embedded Software, 2005, pp. 290–298.
[9] Standard Universal Verification Methodology, available at: http://accellera.org/downloads/standards/uvm, accessed 25.05.2020.
[10] Kamkin A., Chupilko M. A TLM-based approach to functional verification of hardware components at different abstraction levels. Proc. of the 12th Latin-American Test Workshop (LATW), 2011, pp. 1-6.
[11] Petrochenkov M., Stotland I., Mushtakov R. Approaches to Stand-alone Verification of Multicore Multiprocessor Cores. Trudy ISP RAN/Proc.ISP RAS, vol. 28, issue 3, 2016, pp. 161-172. DOI: 10.15514/ISPRAS-2016-28(3)-10.
[12] Averill M. Law, W. David Kelton. Simulation Modelling and Analysis. 3rd edition. McGraw-Hill Education, 2000, 784 p.
[13] TLM-2.0.1. TLM Transaction-Level Modeling Library, available at: http://www.accellera.org/downloads/standards/systemc, accessed 25.05.2020.
[14] 1800-2017 - IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language, available at: https://standards.ieee.org/standard/1800-2017.html, accessed 25.05.2020.

Information about authors / Информация об авторах

Dmitry Alexeyevich LEBEDEV earned a specialist in electronics diploma in 2014 at MEPhI. His area of research interests includes the verification methods study of communication system controllers, interrupts systems and memory subsystems devices with support of protocols of coherence.

Дмитрий Алексеевич ЛЕБЕДЕВ защитил диплом специалиста в области электроники в 2014 г. в НИЯУ МИФИ. Область его исследовательских интересов включает исследование методов верификации контроллеров связи, систем прерываний, устройств подсистемы памяти с поддержкой протоколов когерентности.

Vitaliy Nikolaevich KUTSEVOL received his master's degree in 2013 from MIPT. Area of his scientific interests includes system verification, development of cycle-accurate simulators and event models.

Виталий Николаевич КУЦЕВОЛ получил степень магистра в 2013 г. В МФТИ. Область его научных интересов включает системная верификация, разработка потактовых симуляторов и событийных моделей.

DOI: 10.15514/ISPRAS-2020-32(2)-6

Implementation of Memory Subsystem of Cycle-Accurate Application-Level Simulator of the Elbrus Microprocessors

[1,2]*P.A. Poroshin, ORCID: 0000-0003-0319-5184 <poroshin_p@mcst.ru>*
[3]*D.V. Znamenskiy, ORCID: 0000-0001-8026-9074 <znamen_d@mcst.ru>*
[1,3]*A.N. Meshkov, ORCID: 0000-0002-8117-7398 <alex@mcst.ru>*
[1]*INEUM, 24, Vavilova st., Moscow, 119334, Russia*
[2]*Moscow Institute of Physics and Technology (National Research University)*
9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russia
[3]*MCST, 1, Nagatinskaya st., Moscow, 117105, Russia*

Abstract. Performance characteristics of any modern microprocessor largely depend on its memory subsystem. Naturally, the memory subsystem software model is an important component of the cycle-accurate simulator, and its validity and quality have high impact on the overall accuracy of the simulation. In this paper the cycle-accurate application-level simulator of the Elbrus microprocessor family is introduced. The structure of the cycle-accurate simulator is briefly explained. After that the software model of memory subsystem and its integration as a part of the cycle-accurate application-level simulator are described. We evaluate accuracy of the application-level cycle-accurate simulator on the SPEC CPU2006 benchmark and analyze the simulation errors. Finally, a brief comparison of different Elbrus architecture simulators is given.

Keywords: Elbrus Architecture' Memory Subsystem; Cache Memory; Cycle-Accurate Simulator; Microprocessor; Application-Level Simulation; SPEC CPU2006

For citation: Poroshin P.A., Znamenskiy D.V., Meshkov A.N. Implementation of Memory Subsystem of Cycle-Accurate Application-Level Simulator of the Elbrus Microprocessors. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 61-80. DOI: 10.15514/ISPRAS–2020–32(2)–6

Реализация подсистемы памяти в рамках потактово-точного симулятора уровня приложений микропроцессоров архитектуры «Эльбрус»

[1,2]*П.А. Порошин, ORCID: 0000-0003-0319-5184 <poroshin_p@mcst.ru>*
[3]*Д.В. Знаменский, ORCID: 0000-0001-8026-9074 <znamen_d@mcst.ru>*
[1,3]*А.Н. Мешков, ORCID: 0000-0002-8117-7398 <alex@mcst.ru>*
[1]*ПАО «ИНЭУМ им. И.С. Брука», 119334, Россия, г. Москва, ул. Вавилова, д. 24*
[2]*Московский физико-технический институт,*
141701, Московская область, г. Долгопрудный, Институтский пер., 9
[3]*«МЦСТ», 117105, Россия, г. Москва, ул. Нагатинская, д. 1, стр.23*

Аннотация. Производительность современных микропроцессоров существенно зависит от устройства их подсистемы памяти. Таким образом, программная модель подсистемы памяти является ключевым компонентом потактово-точных симуляторов, и качество этой модели в значительной степени определяет итоговую точность моделирования всего микропроцессора. Данная статья посвящена потактово-точному симулятору уровня приложений, специализированного на моделировании микропроцессоров архитектуры «Эльбрус». В статье дано описание общей структуры

рассматриваемого потактово-точного симулятора. Вслед за этим описаны программная модель подсистемы памяти и особенности ее интеграции как части потактово-точного симулятора. Далее изложены результаты оценки точности разработанного потактово-точного симулятора на наборе тестов SPEC CPU2006 и проведен анализ ошибок моделирования. Завершает статью сравнение производительности симуляторов микропроцессоров «Эльбрус» различных типов.

Ключевые слова: архитектура "Эльбрус"; подсистема памяти; кэш-память; потактово-точный симулятор; микропроцессор; симулятор уровня приложений; SPEC CPU2006

Для цитирования: Порошин П.А., Знаменский Д.В., Мешков А.Н. Реализация подсистемы памяти в рамках потактово-точного симулятора уровня приложений микропроцессоров архитектуры «Эльбрус». Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 61-80 (на английском языке). DOI: 10.15514/ISPRAS–2020–32(2)–6

1. Introduction

As complexity of modern microprocessor and compiler optimizations increases, it becomes almost impossible to predict performance of application without actually running it. This is why cycle-accurate simulators are important, especially during development of hardware.

To be useful, a cycle-accurate simulator should give reasonable approximation of behavior and timings of real hardware, therefore it should be reasonably accurate. And a proper simulation of memory subsystem is a major factor in overall accuracy of simulator.

Due to high complexity of the simulator itself, it is not only important to have a working simulator, it is also important to have means to debug and evaluate it.

In this paper we describe our approach to integration of memory subsystem model into application-level cycle-accurate simulator of Elbrus microprocessors and give an overview of the tools that we developed to help to improve accuracy of this simulator.

The remainder of this paper has the following structure. Section 2 gives brief overview of Elbrus microprocessor architecture, its memory subsystem characteristics and pipeline specifics. Section 3 describes synchronous part of pipeline model and its interaction with memory subsystem model. Section 4 describes the general design of memory subsystem model. Section 5 gives an overview of available debugging facilities for the simulator being developed. Section 6 is dedicated to the simulator evaluation from the standpoint of its accuracy and performance. Section 7 is dedicated to related work about simulator validation. Section 8 gives concluding remarks and briefly describes our plans for further work.

2. Prerequisites

2.1 Elbrus Architecture

Elbrus microprocessors have ISA (Instruction Set Architecture) of VLIW (Very Long Instruction Word) type. In such architectures, performance is achieved through the use of «Wide Instructions» (WI). Each WI consist of several sub-operations, which are executed by hardware in parallel.

Elbrus ISA implies in-order execution. Extraction of ILP (Instruction Level Parallelism) from a program algorithm is the responsibility of an optimizing compiler. This allows to reduce complexity of the hardware.

Each WI of Elbrus ISA can contain several arithmetic, logical operations or memory access operations, control transfer (CT) operations, operations on predicates and others.

2.2 Memory subsystem

Since Elbrus is a VLIW architecture, the memory subsystem accordingly has high ILP potential. Up to 4 memory load or 2 memory store operations can be placed in a single WI.

The cache hierarchy of the Elbrus microprocessors memory subsystem includes Instruction Buffer (IB, responsible for code fetching and generally acting as an instruction cache), data caches (L1D, L2, L3) and various address translation structures (ITLB, DTLB).

Additionally, Elbrus microprocessors have Array Access Unit (AAU), which is used for programmable asynchronous data prefetch, and CLW, which is used for automatic cleanup of data on stack. Both of them asynchronously generate requests to memory subsystem. There are also other components (for example TLU) that insert specific memory requests into synchronous memory access operations stream.

2.3 Pipeline Specifics

Pipeline of Elbrus microprocessors consists of the following stages (from early to late stages from the perspective of individual instruction): L, A, F0, F1, S, D, B, R, E0, E1 etc. First of them (from L to S) are in responsibility of Instruction Buffer. Stage R roughly corresponds to reading of operands, and stages starting from E0 are general execution stages, including arithmetic, address calculation for loads and stores, etc.

Instruction Buffer incorporates several parallel code fetching pipelines. Special control transfer preparation (CTP) operations can be executed to start fetching code for the upcoming control transfers using dedicated preparation pipelines, and a prepared control transfer operation can switch the main pipeline to the chosen preparation pipeline. This mechanism is used for hiding latency of some control transfers.

Several types of pipeline stalls can be distinguished. Each stall type induces different pipeline reaction.

The first stall type is *regular*. The stalled WI is not progressing on the pipeline until the stall condition is resolved.

The second type of stalls includes *B-Stalls* and *L-Stalls*. The most frequent reason for such stalls are unavailable operands. Pipeline logic detects such situations with some delay, that is why simply stopping pipeline progression for the instruction is not enough. Instead, current results of instruction are discarded and for the next ticks instructions are transferred back to the R stage (from E2 for B-Stall and from E0 for L-Stall) until all affected instructions are placed back into the proper order. There is special pipeline logic for cases when one such stall happens during another one. In most cases, one round of B-Stall effectively adds 4 ticks and L-Stall adds 2 ticks of latency.

Even if only one operation of WI triggers stall condition, the whole WI is affected.

When WI passes stage E2, it cannot be stalled anymore and effectively continues its execution without interruptions.

3. Synchronous Pipeline Model

3.1 Overall Design

The cycle-accurate simulator described in this paper is based on functional application-level simulator of Elbrus microprocessor and shares most of the code base with it.

Both the architecture state and the algorithmic behavior are handled by the functional component of simulator. This includes decoding instructions, maintaining state of memory and registers, simulating effects of executed instructions, emulating system calls, etc.

Also, the cycle-accurate simulator reuses some of the facilities of functional simulator, such as internal cache of decoded instructions [1], logging system, command line options parsing, events system, various configuration mechanisms, etc.

To achieve higher simulation performance and at the same time obtain desirable levels of accuracy and maintainability we implement pipeline model, described in [2] as "Hybrid" pipeline model. General idea of this approach is to simulate timing behavior of instructions by as large continuous

chunks of logic as possible, potentially with use of some educated predictions about future behavior. For inevitable situations when predictions are false, simulator maintains additional data for making necessary corrections and does not allow propagation of incorrect speculative behavior to irreversible state.

For example, we assume by default that operations will not be stalled and on this basis we speculatively register availability of the results of the operation according to this assumption. But in case if the operation is actually stalled, the simulator corrects the earlier made assumption and recalculates the moment when results of the operations will be available.

In our case large chunks of timing logic correspond to continuous sequences of pipeline stages that are executed without interruptions.

The algorithm of simulation loop can be summarized by the following steps:

- for the new instructions, do simulation of its algorithmic behavior using functional component of the simulator, saving necessary info for cycle-accurate part of simulation in the process;
- place this instruction into the pipeline model and pass necessary additional info about its execution;
- iterate through each pipeline stage and for each stage determine which instruction is at this stage. If it is the first stage of continuous uninterrupted sequence - speculatively simulate all stages of this sequence;
- update pipeline state (which instructions are at which stage), taking into account possibly occurred stalls.

3.2 Support of Memory Subsystem

Memory subsystem plays a major role in overall performance of the system. It is important to accurately simulate its effects.

Memory subsystem of Elbrus microprocessors is rather complex, so instead of implementing its model from the ground up we adopted the model described in [3]. From the perspective of the pipeline model, this memory subsystem model is regarded as black box with clear but limited interface. This helps us to achieve higher levels of modularity and limit influence of design of cycle-accurate simulator on memory subsystem model so it can be used in several projects more easily.

Model of memory subsystem implements IB (Instruction Buffer), all data caches (L1D, L2 and L3) and MC (Memory Controller). Functional component of the simulator does not directly interact with the memory subsystem model, and the cycle-accurate component directly interacts with IB and L1D by regularly (each tick) forming and passing input to them.

Because the described simulator is application-level, there are no OS effects and no proper memory management. The consequence of this is that there is effectively no virtual address translation takes place during simulation, and memory subsystem functions as if all memory accesses are physically addressed.

General architecture of the cycle-accurate simulator is presented on Fig. 1.

3.2.1 Memory Access Operations

As rather isolated component of the simulator, memory subsystem model does not implement speculative features of the pipeline model. Moreover, because its high complexity it does not some of the stalls (specifically, L-Stalls that affect stages R and E0, and B-Stalls that affect stages R, E0, E1 and E2). These facts should be taken into consideration during integration of the model into cycle-accurate simulator.

Memory subsystem model is designed as cycle-by-cycle model and expects that its main simulation step should be executed once every simulation tick. In our approach this happens in the

main simulation loop body of the pipeline model. From the perspective of the pipeline model, the specific moment when this can happen must satisfy following conditions.

- It must happen before other operations that are currently on the pipeline check availability of operands, otherwise there would be excess stalls. This check happens during simulation of the R stage.
- It must not happen after the most recent instruction with memory access operations (which should be send to memory subsystem model next) irreversibly (non-speculatively) reached earliest stage of possible feedback from memory subsystem that should be immediately acted upon. This corresponds to stage E2 when earliest possible data return from L1D cache can happen, and which should be taken into account during checking availability of operands on the same tick.

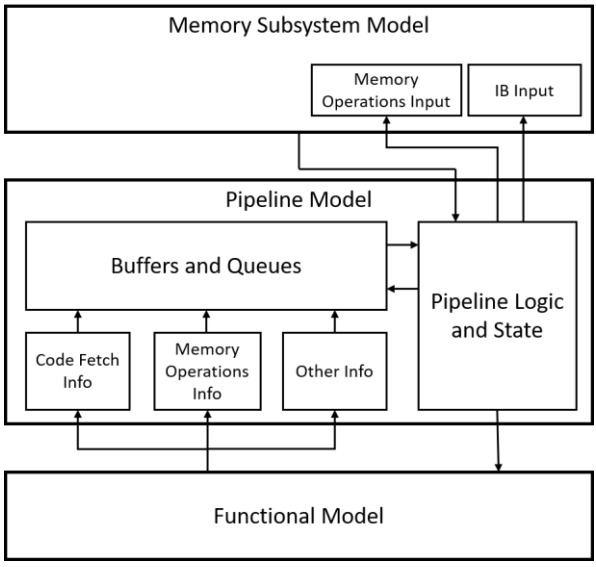

Fig. 1. General architecture of described cycle-accurate simulator

This means, at least from perspectiveof the pipeline model, that it is possible to send new memory access operations and execute memory subsystem simulation step anywhere between simulating (non-speculatively) stages R and E2 of instruction. But from the perspective of memory subsystem model, the earlier it can start processing new operations, the better, as this gives to the model more simulation steps to process each individual operation and probably will require less changes to original memory subsystem model. So in our final approach the information about memory access operations of instruction is sent and simulation step of memory subsystem model is executed just after processing stage E0 of this instruction just before processing stage R of other instructions.

Because memory subsystem model does not support "speculative" features of the pipeline model, and there is no means to make corrections of the model's state, new memory access operations should be sent to model only when it is guaranteed that there will be no corrections that can change the data related to these operations. In our current model, the information about memory access operations is mostly determined by the functional component of simulator, which is not affected by "speculative" features of the pipeline model, so this requirement is fulfilled.

Other aspect to consider is the lack of support of rollback during some stalls. This means that in case of rollback of the instruction actions, sending of the memory access operations of this instruction to the memory subsystem model should be postponed until the next opportunity after the rollback. This also means that rollbacks should be known with certainty before (or at least at) stage E0, which is true for our current model.

Information about memory access operations is gathered during functional simulation at the start of processing of instruction. This information includes type of operation, memory address, destination register number (for load operations) and various other attributes. After functional simulation, this data is saved in the pipeline model alongside with other information about instruction and is used to form request to memory subsystem model when the time comes.

There can be several memory operations in-flight at the same time, and to distinguish them each memory access operation is associated with a unique token (internally represented by an integer). Because memory access operations can take many ticks to complete, response from the memory subsystem model can arrive when relevant instruction is no longer present in pipeline model (all instructions that pass stage E2 are deallocated from the pipeline model as there are no more stalls that can affect it and most of the timing behavior is certain and speculatively simulated). To address this, pipeline model implements additional buffers that hold the information necessary for proper action on response from the memory subsystem.

3.2.2 Code Fetching

Memory subsystem model also simulates the Instruction Buffer, which handles fetching of instructions from memory.

Process of instruction fetching is divided into several pipeline stages, but for simplicity they are not fully represented in the pipeline model. Pipeline model uses a special pseudo stage (internally named F) instead. New instruction is placed on this stage after its functional simulation. At the same time, on the same simulation tick the information about this instruction (mainly its address and size) is passed to the memory subsystem model. Instruction can leave pseudo stage F only after there was a response from the memory subsystem model about successful completion of the code fetching.

Some of the Elbrus control transfer instructions are separated into two parts: "control transfer preparation" (CTP) and "control transfer" (CT) instructions. CTP instructions initiate fetching of the code for the new path (which happens in parallel with regular fetch of the current path), and CT instructions implement control transfer to this code. This functionality allows to reduce latency of branches in some cases. This fetch is also a responsibility of IB and is simulated by the memory subsystem model. Information about CTP operations is gathered during functional simulation and passed to the memory subsystem model in the beginning of processing of the instruction by the pipeline model (at pseudo stage F). When corresponding CT operation is executed, address of target instruction is passed to the memory subsystem model by standard mechanism (described earlier), and if IB has not fetched target instruction yet, then there will be a pipeline stall (at pseudo stage F).

3.2.3 Hardware Generated Operations

In some cases, hardware of Elbrus microprocessor issues special instructions (later referred to as "hardware instructions") that are not directly represented in binary code. Most notable of such hardware instructions are instructions which spill and fill register file contents to/from memory when active register window changes (for example, after procedural control transfers). Most of the hardware instructions perform memory accesses, and therefore play an important role in interaction of synchronous pipeline and the memory subsystem.

The functional simulator already simulates hardware instructions, as the triggering conditions for them are part of instruction set, and it is necessary to account for them to correctly maintain microprocessors visible state. But functional simulation of hardware instructions is significantly simplified, as the whole process takes place during simulation of ordinary instruction with a triggering condition occurring during one of the simulation steps (while one spill of register file can consist of many write accesses to memory).

Although functional simulation of hardware instruction is inaccurate from a timing standpoint, cycle-accurate simulation reuses most of it. During functional simulation information about all execution of hardware instruction is saved and passed to the pipeline model alongside with the instruction which triggered them. When this instruction reaches specific stage the saved information about hardware instructions is used to appropriately insert them into pipeline instead of next ordinary instruction. This continues until all hardware instructions are inserted.

It is possible to save information about hardware instructions in compact way. For example, although each spill (or fill) of register file usually consists of a sequence of many individual instructions, algorithm to generate them is predetermined and defined by several parameters. Also, all of the memory accesses of each individual spill (or fill) sequence share most of the attributes.

While ordinary instructions enter and leave the pipeline model in a FIFO fashion (enter at pseudo stage F and leave after stage E2), this is no longer true in the presence of hardware instructions. They are not fetched from memory (as they are not directly represented in binary code) and enter pipeline at stage R. This was not considered during original design of data structures of pipeline model and necessary changes were implemented.

4. Memory Subsystem Model

4.1 Adaptation

For usage inside the application level (AL) cycle-accurate Elbrus architecture simulator, the memory subsystem timing model, which originated from the system-level memory subsystem (SLM) cycle-accurate simulator [3], underwent several changes.

According to the pipelining mechanics described above, code fetching from the IB cache and the instruction-word related memory accesses are processed on different pipeline pseudo stages. The timing model engine code was distributed between those pseudo stages, and separate input request queues for the code fetching stream and the memory access operations, with respective pipeline stalls, were created. It is worth mentioning that both parts of the model continue influencing each other through the “lower” part of the memory subsystem.

The precise WI data dependency checking and conflict resolution logic was implemented as a part of the core pipeline timing model. That is why a simplified scoreboarding mechanism utilized in the SLM cycle-accurate simulator was replaced by a set of fast callbacks, which report the memory operation processing status to the core pipeline timing model (e.g., a cache hit or a cache miss, MU internal queues overflow, etc).

The virtual address translation mechanics is not used for the AL cycle-accurate simulator. All virtual addresses are equal to physical addresses, and all MMU mechanics including page table search, ITLB and DTLB lookup are disabled. In addition, the L1D cache physical tags lookup mechanism, which is used as an anti-aliasing technique for virtual address cache indexing, is turned off. This gives an opportunity to simplify the memory subsystem model code and to increase its performance.

4.2 Improvements

Besides the adaptation changes described above, we significantly improved the memory subsystem model in terms of accuracy.

Multiple enhancements of IB, L1D, L2 and L3 cache models including a multitude of refinements for caches algorithms, latency calibrations and buffer depth adjustments were made.

For the shared L3 cache of a multicore processor, even for a single-threaded workload attached to a specific core, one should also consider influence of idle processor cores. Each of the idle-cores generates a sparse stream of memory accesses to the L3 cache. The combined idle-core stream can have a sufficient magnitude to interfere with the application core stream or at least to influence L3

cache performance metrics. To emulate the described background idle-core stream for the single-core configuration of the simulator a synthetic adjustable L3 cache access stream was introduced. This mechanism partially compensates idle-core traffic, and its precise adjustment is yet to be done.

Workloads like 410.bwaves from the SPEC CPU2006 benchmark [4] show strong performance dependency on the system memory characteristics like latency or throughput, which in turn can vary across different memory access patterns in different workloads. With a view of the simulator's accuracy improvement, a model of DDR4 memory controller (MC), as a part of the memory subsystem timing model, was implemented. Current SLM and AL simulator configurations, which correspond to an 8-core Elbrus processor, include four MCs with the original interleaving scheme. The MC model was simplified for the sake of performance increase at the cost of some potential accuracy loss. Even with this tradeoff, adding MC model into the AL cycle-accurate simulator yielded additional 4% overall accuracy improvement on the SPEC CPU2006 benchmark, with negligible simulator speed degradation.

5. Profiling and Debugging Facilities

It is important to have debugging tools in simulator, both for users of the simulator and for its developers. As more and more features are implemented (most notably accurate model of memory subsystem), complexity of possible mistakes in simulator rises, which demands for more advanced internal debugging capabilities of the simulator.

The simulator described in this paper provides two main debugging capabilities, namely execution log and execution statistics (profiling). Their design is described in this section.

5.1 Execution Cursor

Both execution log and execution statistics need the way to identify different events in simulator. For this purpose, the concept of «execution cursor» is implemented in pipeline model.

Content of the execution cursor should point to the specific moment of simulation process, preferably both in terms of internal phases of simulation and in more simulator independent terms. In our implementation execution cursor specifies execution tick (when event should have happened on real hardware), simulation tick (when event was simulated, with respect to «speculative» nature of pipeline model), pipeline stage, IP of instruction, fetch tick of instruction, specific operation of instruction and internal simulator's description of operation.

Current execution cursor should be accessible from (almost) everywhere in the code. To achieve this «current» execution cursor implemented as a field of pipeline model object, which is accessible almost everywhere.

Logically execution cursor should have different structure for different events and parts of the simulation path. For example, some events (like pipeline stall on code fetching) should be attributed for whole instruction, but specifying one operation is meaningless for them. But for simplification and performance purposes, in our implementation the execution cursor always specifies whole range of attributes, possible with garbage values, but its users (various code pieces of the simulator) know which part of it is valid and which is not. For example, when a pipeline stall happens on fetching and appropriate counter of execution statistics is going to be incremented, it is known that operation related attributes of the execution cursor are undefined.

Also, the execution cursor should function like a stack, since most of the simulation process is stack-like (whole pipeline is at the bottom of the stack and specific instructions and operations are at the top). When simulation moves higher into the stack, relevant attributes of execution cursor should be updated, and when simulation returns back, the execution cursor should be restored as earlier updates are no more relevant. For performance reasons, restoration of execution cursor state is not implemented, as most of the time updated attributes are not valid in lower parts of

simulation stack, so no information is lost. In a few cases where information is lost during update, saving and restoration of the execution cursor are implemented manually.

5.2 Execution Log

The main function of execution log is to present a detailed view of execution process and of microprocessor state evolution with respect to timing. Execution log includes information about ticks when each executed instruction reached certain pipeline stage, whether it was stalled, its stall reasons and etc. This information can be used to investigate performance anomalies and unexpected behavior.

Due to «speculative» features of the pipeline model, the execution log can not simply reuse standard logging facilities of functional simulator. For example, if availability of some register was speculatively but incorrectly updated, it would be wrong to include this update in the log. Therefore logging of any such event should be postponed until its correctness is certain. To support this, any intention to log an event is placed in execution log queue, indexed by tick number when event should actually (non-speculatively) happen. This queue is inspected every simulation step, and every event that is indexed by current non-speculative tick («tick of no return») and is still in the queue, is printed. If some speculation was wrong and a correction of speculative state is needed, then the queue of execution log is scanned and all wrong events are erased.

When an event is placed in the queue, current execution cursor is examined and all its information relevant for logging is placed in the queue alongside with the event. For example, when availability of a register changes, the information about current operation (which, in fact, is the reason for this event happening) is saved in the queue for a more informative output.

5.3 Execution Statistics

Other useful functionality of the simulator is the ability to gather statistics of execution. While the execution log is useful for analyzing specific moment of execution, execution statistics provide more convenient view of integral characteristics of the whole execution process. In other words data of the execution log is spaced in time domain and data of execution statistics are spaced in code domain.

As a tool, execution statistics collection is implemented as two main parts:

- facilities (internal for cycle-accurate simulator) for accumulation and counting of interesting events and for forming (raw) output with accumulated data.
- facilities for processing and transforming (raw) output with statistics into a more human readable format.

5.3.1 Accumulation of Statistics

Execution statistics collection in cycle-accurate simulator is represented as an in-memory map indexed by addresses of instructions (IPs). Each entry of this map contains some general counters for this IP (like number of executions, number of stalls etc.) and a map of event counters, indexed by event type and attributes.

All of the events that are currently being collected by simulator are always certain and can not be speculative, unlike events of the execution log. This allows to simplify current implementation of execution statistics, but it is always possible to apply solution from execution log implementation, if necessary.

For each IP, execution statistics accumulate the following information:

- total number of executions, stalls, NOPs etc.;
- number of stalls of each type (for example, stalls caused by unavailable data in register file and bypasses, stalls caused by fetching of instructions, etc.);
- number of procedural control transfers;

- number of non-procedural control transfers;
- some other statistics.

It is important to properly count pipeline stalls. For example, instruction can be stalled at stage B, but not because it has some reason itself, but because currently B-Stall (or L-Stall) is active, which blocks pipeline progression for instruction on stage B. Counting such stall not only misattributes stall for wrong instruction, but also counts effects of real reason for stall (reason of B-Stall) twice.

It can be useful to separate stalls not only by their general type, but also by some additional factors. For example, for every stall caused by unavailability of data in register file, there is a producer of this data and a consumer of this data, which is actually stalled. For this reason, statistics counters are indexed not only by type of event, but also by additional attributes, such as specific channel of instruction that was stalled, IP and channel of instruction that produces the result that leads to stall etc. For different types of events different types of additional attributes are used.

Counter of (non-)procedural control transfers are specific for each target. These counters are later used for construction of call graph and calculation of approximate inclusive costs of functions. Non-procedural control transfers are also essential for this, as some compiler optimizations (such as tail call optimization) transform procedural transfers into non-procedural ones. Information about non-procedural control transfers helps to detect such transformations.

5.3.2 Processing of Statistics

Although raw output of execution statistics is text based and can be analyzed directly without any additional processing, it is meant to be transformed into a more human readable format with some tools external to the simulator.

For this purpose, a separate tool was implemented as a script with the following possible output options:

- disassembly annotated with counters from raw output of execution statistics;
- summary of counters, aggregated for functions (with non-inclusive aggregation across instructions of function body and with inclusive aggregation across instructions of all callers);
- summary in the format compatible with the profile visualization tools such as KCachegrind [5].

Unprocessed execution statistics do not contain function names or disassembly. They are supposed to be obtained with objdump utility with support of Elbrus ELF files from binary executable file of the program being investigated.

The first step of execution statistics processing is its parsing and converting into a representation more suitable for further transformations. Then, a call graph is constructed, if necessary for chosen output.

As raw execution statistics for each control transfer counter contain only IPs of source (caller) and target (callee) of the transfer, it is impossible to construct a call graph which considers full call stack at the moment of transfer. To address this, during construction of a call graph a simple heuristic is used, which assumes that for each function all executions of its body are similar for all real call stacks. This assumption allows to calculate inclusive costs of functions by evenly distributing all counters of a callee across all individual calls from callers, starting from leaves of the call graph.

This method of inclusive function costs calculation does not work if a call graph has cycles, which are usually caused by recursive calls. To combat this, before calculation happens all cycles (more specifically, all strongly connected components) are detected and converted into special aggregated nodes of the call graph.

During construction of a call graph, not only procedural control transfers are considered, but also non-procedural ones. But not all non-procedural control transfers should be taken into account, as

most of them are truly logically non-procedural. In our implementation, only non-procedural control transfers that cross function boundaries are converted into calls.

For visualization of execution statistics, we use KCachegrind program. It is mainly used for visualization of output from callgrind utility, but the input format is simple and flexible enough for visualization of other call graph like data. Our script supports output in KCachegrind compatible format with only aggregated information about functions and with more granular information about individual instructions. KCachegrind was built in support for viewing annotated disassembly, but not for the Elbrus disassembly. To achieve similar functionality, we utilize view of annotated source code, but in place of source code we use disassembly manually obtained with objdump utility with support of Elbrus binary format.

5.4 A Validation Case Study

As an illustration of the introduced simulator statistics usage, the authors present a simulator validation case study. During one of the validation iterations, a significant run time mismatch for the 444.namd workload (taken from SPEC CPU2006 benchmark) was found: the AL cycle-accurate simulation duration result was increased by 17%, while the cache hit rate differences stayed tolerable. The tasks' reference system and simulator profiles were collected using perf Linux profiler and the described AL simulator's statistical tools and methodology, respectively. The key profiling parameter was the total number of pipeline stalls. The profiles are shown in listings 1 and 2.

```
12,05%  ComputeNonbondedUtil::calc_pair_fullelect
11,45%  ComputeNonbondedUtil::calc_pair_energy_fullelect
 9,45%  ComputeNonbondedUtil::calc_pair
…
```

Listing 1.444.namd perf-generated profile, reference machine

```
13,19%  +9,45%  _ZN20ComputeNonbondedUtil19calc_pair_
                 fullelectEP9nonbonded <0x38a30>
13,00% +13,57%  _ZN20ComputeNonbondedUtil26calc_pair_
                 energy_fullelectEP9nonbonded <0x448a0>
 9,53%  +0,79%  _ZN20ComputeNonbondedUtil9calc_pair
                 EP9nonbonded <0x20a98>
…
```

Listing 2.444.namd simulator-generated profile with relative function weight difference

```
5.02 |3ff78:
     |…
     | faddd,sm %db[94], %db[103],
%db[62]
     |…
...
     |40020:
     |…
     | stgdd %r46, 0x0, %db[62]
```

Listing 3.444.namd perf-annotated disassembly, reference machine

```
0x3ff78                                          # E=20510404 S=10408678
                                                   N=0 HW=0 LD=0 ST=0
  …                                              # IB_FETCH_NOT_READY = 178
  faddd,sm %db[94], %db[103], %db[62]            # RF_REG_NOT_READY
                                                   <source IP 0x3f728> = 88
  …                                              # RF_REG_NOT_READY
                                                   <source IP 0x3fb00> = 116
                                                 # RF_REG_NOT_READY
                                                   <source IP 0x40428> = 877024
                                                 # RF_REG_NOT_READY
                                                   <source IP 0x40450> = 9531272
…
0x40020                                          # E=20510404 S=66987408
                                                   N=0 HW=0 LD=0 ST=19440799
  …                                              # RF_REG_NOT_READY
                                                   <source IP 0x3ff78> =
66987408
```

Listing 4.444.namd simulator-annotated disassembly

For the AL cycle-accurate simulator the relative weight of the hottest procedures calc_pair_fullelect and calc_pair_energy_fullelect has increased by 9% and 14%, respectively. Taking a closer look at the annotated disassembly of the hottest procedure calc_pair_fullelect revealed a significant stall increase (almost up to 9%) for an instruction at the 0x40020 address, which executed absolutely seamlessly on the reference machine. The perf-generated and the simulator-annotated disassemblies are shown in listings 3 and 4.

As we can see, the data for the store operation (%db[62]) in the 0x40020 instruction word is produced by floating-point addition instruction located at 0x3ff78. The consumer (store at 0x40020) is scheduled for execution with a 4-cycle latency after the operand producer (faddd at 0x3ff78), but it gets stalled very frequently because faddd's result is not ready. Notably, on the real hardware the consumer instruction does not stall at all. The authors checked the CPU's scheduling rules, which are a part of the Elbrus ISA. It turned out that the result availability latency of type {faddd → integer / memory} is 6 cycles, but for the special case {faddd → store data}, the latency is 4 cycles. Thus, the complier created an optimally scheduled code. Further examination revealed that this very special case was not taken into account by the simulator's logic. Fixing this inaccuracy shrinked the total run time error of the 444.namd workload to a negligible value (below 1%).

6. Evaluation

6.1 Simulator Accuracy

Accuracy is one of the key simulator quality markers in a sense of reference and modelled system characteristics convergence. It can vary significantly across different classes of workloads (for instance, integer or floating-point workloads), and, moreover, across different workloads of the same class. As for physical microprocessor systems, for the cycle-accurate microprocessor simulator characteristics evaluation, a standard benchmark, which includes a variety of workloads from different classes and problem domains, should be used. The authors chose the standard cross-platform SPEC CPU2006 benchmark, which has been used for the Elbrus architecture microprocessors performance evaluation [6][7]. We compiled SPEC CPU2006 workloads with the Elbrus compiler toolchain developed by JSC MCST. It is essential that, for the sake of cycle-accurate simulator compatibility, some optimization features were disabled at compile time. For example, the usage of Array Access Unit was disabled because the latter is not yet appropriately supported in the timing model of the simulator. In other words, all the reference system and the

simulator data presented below correspond to some intermediate performance mode sufficient for the simulator validation, but not to the peak performance mode.

The authors used a physical machine based on 8-core Elbrus CPU as a reference system for the AL cycle-accurate simulator accuracy benchmarking. The configuration of the reference system is presented in Table 1.

For most of the SPEC CPU2006 tasks the test input data set was used, and a smaller fraction of workloads was run with the train data set. This choice of input data sets was motivated by a reduction in simulator run time for those workloads, which in turn allowed the authors to speedup the simulator validation workflow loop that requires frequent task re-execution.

Fig. 2 and 3 present the main accuracy evaluation results for the AL cycle-accurate simulator, namely the task run time normalized by the run time on the physical reference system (Fig. 2), and the workload error distribution (Fig. 3). Workloads on Fig. 2 were run with the test input except workloads where the train input data is explicitly specified. The geometric mean (GM) is presented for benchmarks that include multiple runs. These results show that for more than a half of the workloads the error stays inside a 4% limit, where the CINT tasks have, on average, a lower error in comparison to the CFP tasks. The vast majority of the tasks has a 12% upper bound with several outlying cases with 25% error at worst. The geometric mean for the error on the whole SPEC CPU2006 set is below 5%, which is an admissible result [8][9].

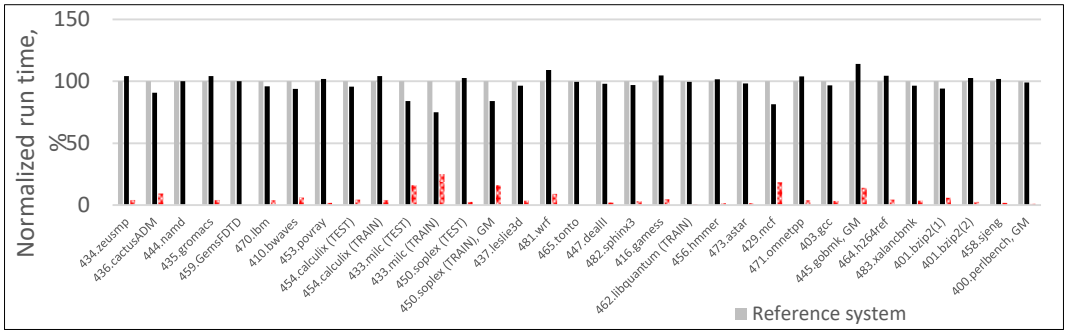

Fig. 2. SPEC CPU2006, AL cycle-accurate simulator accuracy

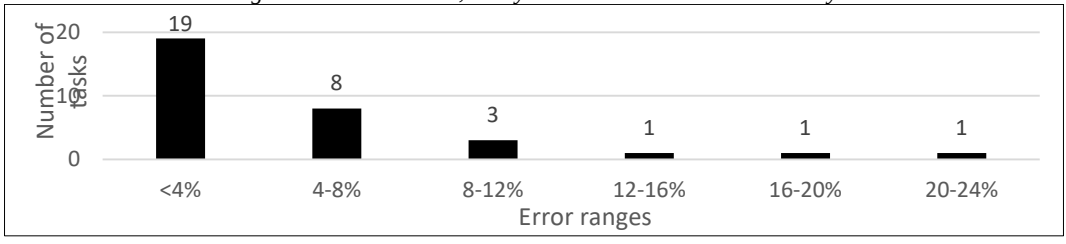

Fig. 3. SPEC CPU2006 workload error distribution

Table I. Reference system configuration

CPU	Elbrus, 8 cores, 1 node
Clock frequency	1500 MHz
L1 instruction cache (IB)	128 KBytes, 256-byte cache line, 4-way set-associative, virtually addressed
L1D cache	64 Kbytes, 32-byte cache line, 4-way set-associative, virtually addressed
L2 cache	512 Kbytes, 64-byte cache line, 4-way set-associative, 4-banks interleaved, physically

	addressed
Shared L3 cache	16 Mbytes, 64-byte cache line, 16-way set-associative, 8-banks interleaved, physically addressed
RAM channels	4 channels DDR4
RAM size	DDR4-2400, 128 GBytes
Operating system	OS Elbrus based on the Linux kernel
Linux kernel version	4.9.0-4.1-e8c2

Memory subsystem is one of the most contributing sources of pipeline stalls, especially for a VLIW architecture like Elbrus since memory accesses can have sophisticated behavior that can be difficult to be predicted and tackled at compile time. In this context, the hit rate for different levels of the cache hierarchy is an important metric that correlates with the overall system performance, and matching of reference and simulator-originated hit rate values is an important simulator timing validation criterion. Fig. 4, 5 and 6 show hit rates for the L1D, L2 and L3 caches on the reference system and the AL simulator. For private core caches (L1D and L2) there is a high correlation between hit rates on the real and modelled CPU (mean hit rate error is 3% and 2% respectively). The situation is different for the L3 cache, namely the mean L3 hit rate error is noticeably higher (18%).

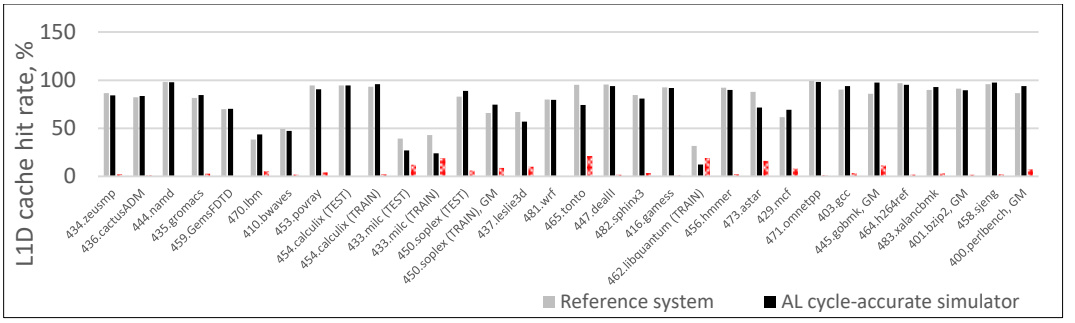

Fig. 4. SPEC CPU2006 L1D cache hit rate

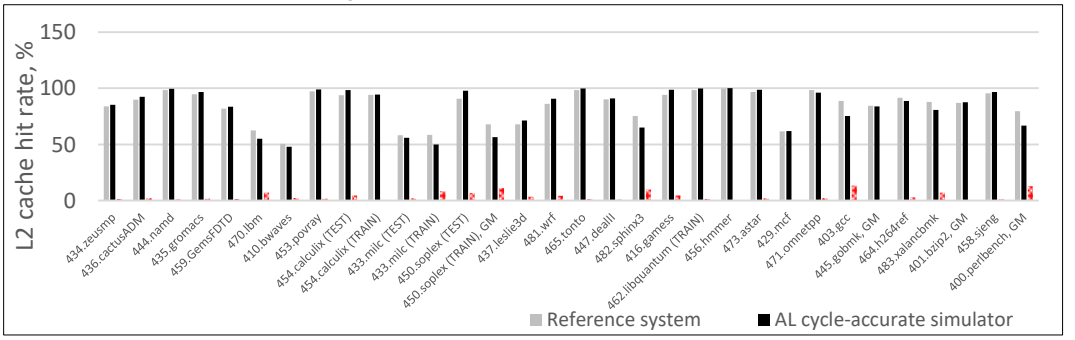

Fig. 5. SPEC CPU2006 L2 cache hit rate

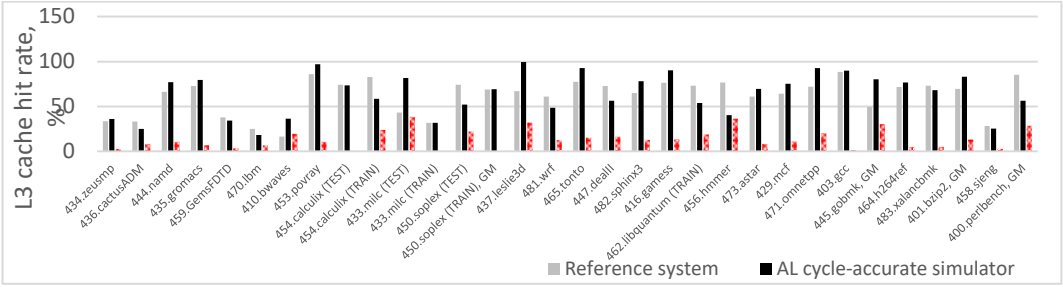

Fig. 6. SPEC CPU2006 L3 cache hit rate

Modelling errors, which result in performance metrics value mismatches described above, can be arranged in a number of groups as a rule of thumb.

- Functional limitations of the cycle-accurate simulator, like hardwired CPU execution mode, simplified system configuration, etc.
- Logical inaccuracy in the timing model. E.g., redundant or insufficient pipeline stalls, wrong hardware algorithms implementation, etc. This group of errors should be diagnosed and fixed during cycle-accurate simulator validation (see 444.namd validation case study above).
- Different performance counters interpretation. As an example of this error group, one can consider a pipelined cache memory with an number of intermediate structures like Store Miss Buffer, Miss Status Hold Registers, etc. In-flight operations can be stored in these buffers in a multitude of internal states, which in turn can be interpreted in a different way by physical and modelled performance counters.
- Incremental cache hierarchy error propagation. If an error appears at some level of the cache hierarchy model, it tends to propagate further down the lower levels. For example, an illegal L1D cache miss causes a spurious L2 cache request, which alters the hole underlying memory subsystem behavior. That is why lower cache levels in the simulator have higher relative variation in request stream intensity and structure (and, as a result, a greater hit rate error) in comparison with higher level caches like the L1D cache.

When analyzing the error rate for the overall accuracy data on Fig. 2, one can notice several outliers with the highest error: 433.milc (test — 16%, train — 25%), 429.mcf (18%), 445.gobmk (14%) and 450 soplex (train – 13%). These cases are worth discussing while keeping in mind the modelling error classification presented above.

For 433.milc workload (both test and train input data), a relatively high DTLB hit rate is common. On the reference system, among the overall 7017 million memory accesses, DTLB misses comprise around 1%, or 70 million misses in total. This exceeds DTLB miss rate of most tasks considered in this paper by several orders of magnitude. Suppose each DTLB miss takes one memory reference for the page table lookup, and it always hits in the L3 cache (which is an optimistic estimate). The L3 cache hit latency takes a few tens of clock cycles, so the cumulative DTLB miss stall penalty will definitely exceed 1000 million clock cycles. The latter number fully explains the difference in real and modelled execution time. This is a simulator functional limitation case, namely the absence of address translation mechanisms and MMU cache lookups in the AL cycle-accurate simulator. One can say that the AL cycle-accurate simulator has both 100% ITLB and DTLB hit rates, which result in the absence of TLB miss-originated pipeline stalls.

For the AL cycle-accurate simulator, the 429.mcf and 450.soplex (train) tasks have a higher L1D hit rate than expected, which could result in erroneous task speedup. In both cases the reasons for the divergence are likely to be simulator logical inaccuracy, which are yet to be fully investigated. Also, there are some functional limitations of the simulator that tend to distort memory subsystem dynamics and complicate the analysis. Namely, on a real machine, the memory allocation is eventually done by the OS kernel memory management mechanism using processor's MMU hardware. Virtual memory is allocated in 4KB, 2MB or 1GB physical memory pages, which are

often grouped into contiguous blocks by the allocation algorithm, where possible. Moreover, some virtual memory pages with distinct virtual addresses can be mapped to the same underlying physical memory page, e.g. for the copy-on-write and zero pages, techniques used by the Linux kernel. The AL cycle-accurate simulator does not reproduce those algorithms. The underlying physical memory is always available, and all physical addresses are identical to virtual, as noted above. This results in effectively different physical addresses footprint for the AL cycle-accurate simulator with respect to the reference system, hence different memory subsystem behavior starting from the L2 cache and below. We found some indirect markers of the limitation's impact. To begin with, we used the page-types Linux utility to get a glance on the virtual-to-physical address mapping of the 426.mcf's task on the reference machine, where only the referenced pages were considered. A physical address heatmap was collected for the cycle-accurate simulator, with physical memory region granularity of 4KBytes. We found that the physical memory footprint on the real machine is far more dense in comparison with the AL simulator, especially when it comes to usage of virtual pages mapped to the same physical page. Also, using aliased virtual pages can cause additional L1D cache misses from the anti-aliasing mechanism, which is not present in the AL version of the cycle-accurate simulator. Secondly, an interesting difference in the L3 statistics for the 429.mcf workload was found. The L3 cache has a special hardware structure for in-flight requests. A hit in this structure occurs when multiple in-flight L3 cache requests access the same cache line. In this situation some additional stalls can occur. The overall L3 request number on the simulator is close to the reference machine (with 20% difference), but the number of hits in the in-flight requests buffer for the reference machine is by three orders of magnitude greater. Thus, frequent physical address collisions on the reference system are not reproduced by the AL cycle-accurate simulator, which indirectly points to L2 cache model inaccuracy or to the address mapping limitation described above, or both.

The 445.gobmk workload with test input consists of several subtasks, which have very diverse numbers of instructions, from tens of millions to billions. «Short» subtasks tend to have high inaccuracy due to the reference machine run time variation and some simulator limitation effects, which are negligible for «long» workloads. Those «short» subtasks have high negative impact on the overall 445.gobmk workload accuracy. For the same reason, the 462.libquantum task was run only with the train input data.

The cache hit rate error data on Fig. 4, 5 and 6 can be also interpreted with the help of the introduced simulator error classification. Generally speaking, the private core caches (L1D and L2) hit rate errors are quite low except a couple of outliers, but the mean shared L3 cache hit rate error is greater. For private core caches (Fig. 4, 5) most high error cases are likely to be performance counter interpretation errors of the memory subsystem timing model (like 462.libquantum for L1D and 482.sphinx3 for L2). The L1D and L2 errors are almost uniformly distributed across workloads, and it is difficult to pick an evident error group for most of them. For the L3 cache the situation is different. Here errors of all four groups severely distort modelling results. Multicore reference configuration has idle core L3 traffic. Moreover, cache coherency mechanisms impact the L3 cache hit rate and alter L3 performance metrics meaning on the reference system. Incremental error propagation uniformly augments error margins across all workloads. Finally, some specific L3 cache model logical inaccuracies exist. For the 410.bwaves task, the hit rate is too high, since this workload is known as a streaming one with low L3 hit rate [6]. Nevertheless, high L3 hit rate error does not always severely influence overall results, since for most tests the memory accesses stream intensity falls when moving to the lower levels of memory hierarchy, especially for workloads that fit in higher level caches.

6.2 Simulator Performance

Running speed is an important cycle-accurate simulator characteristic. It directly affects the simulator validation rate and also the simulator's usability: the speed should be sufficient to run the standard benchmarks common for the modelled architecture. Here we present a brief speed

comparison of the pure functional Elbrus architecture simulator (FUNC), the considered AL cycle-accurate simulator (AL) and the system-level memory subsystem simulator described in [3] (SLM). We conducted comparison on several benchmarks from the CINT and the CFP packages. Those benchmarks are chosen in such a way that their clocks-per-instruction (CPI) parameter varies significantly. We chose the clock-per-wide instruction (CPWI) characteristic measured on the reference system as a simple approximation of the CPI. Modelling results are shown in Table 2.

Functional simulator speed is often expressed in terms of ticks per second, where a tick basically corresponds to an executed instruction. In case of comparison with the cycle-accurate simulators, choosing this performance measurement unit would create some confusion. Instead we expressed the functional simulator's speed in the same performance units as both cycle-accurate simulators' speed, notably in cycles per second, where the number of cycles was obtained from the AL simulator run for a particular benchmark. Table 2 shows that functional simulator's speed is between 5,081 MHz and 10,565 MHz, which is comparable to FPGA processor prototypes. The AL simulator speed varies between 462 KHz and 817 KHz, which is an order of magnitude slower than pure functional instruction set modelling. This is typical since detailed timing models overhead is usually quite severe.For the SLM simulator, the selected benchmarks were run under OS "Elbrus" based no the Linux kernel version 4.19.72. The speed is within the limits of 577 KHz and 1066 KHz. For both considered cycle-accurate simulators the speed tends to increase with the CPWI parameter. This can be explained by simulators' design: during the stall cycles processing, only a smaller fraction of pipeline and memory subsystem model mechanisms are active, and such kind of cycles usually run faster. On average, the SLM simulator is by 13% faster than the AL simulator. For the considered CFP workloads the geometric mean speed difference is 4%, and for the CINT workloads is 21%. The gap gets narrower for lower CPWI tasks (453.povray and 471.omnetpp), and wider for higher CPWI tasks (410.bwaves and 429.mcf). The latter means that the full accurate Elbrus core pipeline timing model overhead of the AL cycle-accurate simulator is greater than the SLM simulator's overhead imposed by the MMU model, lightweight memory scoreboarding and the OS code simulation time combined together. Yet there is some room or optimization (see Section 8). To summarize, simulators' running speeds are quite comparable to each other and acceptable for running the SPEC benchmark package with test and train input data sets: running a 40-billion cycle benchmark, takes roughly a 24-hour period.

Table 2. The AL cycle-accurate simulator and the SL memory subsystem simulator speed comparison

Workload		**CFP**			**CINT**		
		453. povray	**434. zeusmp**	**410. bwaves**	**471. omnetpp**	**462. libquantum**	**429. mcf**
CPWI		1.6	3.1	6.6	1.7	3.5	4.6
Machine		Intel Core i7-2600 CPU @ 3.40GHz, 16GB RAM		Intel Xeon E3-12xx v2 (Ivy Bridge), 32GB RAM			
Simulator speed, MHz	FUNC	5.081	4.152	9.442	3.579	3.361	10.565
	AL	0.796	0.670	0.817	0.462	0.609	0.795
	SLM	0.734	0.699	0.965	0.577	0.733	1.066
Speed (FUNC) / speed (AL)		8.37	8.43	12.67	6.30	5.43	13.77
Speed (AL) / speed (SLM)		1.09	0.96	0.85	0.80	0.83	0.75

7. Related Work

It was demonstrated by [10] that proper validation and calibration of simulator is very important for its accuracy. Accuracy evaluation made during this study showed that the least error in accuracy is achieved by simulators Sniper and ZSim, which have been validated against real hardware. It is also worth mentioning that these simulators are most accurate despite being of application level type.

Simulator Sniper was validated [11] against Intel Nehalem architecture and was shown to achieve average error of 11.1% on the SPLASH-2 benchmarks [12]. This high accuracy preserved (with reasonable variation) during later evaluation conducted in [10] on SPEC-CPU2006 [4] and MiBench [13] benchmarks targeting Intel Haswell architecture.

Simulator ZSim was validated [14] against Intel Westmere architecture. It achieved absolute IPC error of 8.5% on SPEC-CPU2006 [4] benchmark (single-threaded benchmarks) and .11.2% on combination of PARSEC [14], SPLASH-2 [12] and SPEC-OMP2001 [15] benchmarks (multi-threaded benchmarks). Absolute error for Intel Silvermont configuration of simulator is 20.9%, which is roughly twice as large comparing to Intel Westmere configuration. This emphasizes importance of validation and calibration against specific target hardware.

Simulator SiNUCA was validated [9] against Intel Core 2 Duo and Intel Sandy Bridge microprocessors. Validation was conducted on two sets of benchmarks: specially constructed micro-benchmarks and SPEC-CPU2006 suit [4]. Micro-benchmarks were constructed to evaluate specific features of microarchitecture and interactions of different components of microprocessor. On these micro-benchmarks simulator achieves average error of 10% for Intel Core 2 Duo and 6% for Intel Sandy Bridge, while on SPEC-CPU2006 it achieves average error of 19% for both hardware configurations, which is significantly larger than for microbenchmarks. This error increase reaffirms importance of selection of representative workloads during simulator accuracy validation.

8. Conclusion and Future Work

Cycle-accurate simulators are very useful and important class of tools for investigating performance of applications and for exploring and evaluating design space of processor microarchitectures. And memory subsystem simulation is major part of any cycle-accurate simulator.

But, as microprocessors become more and more complex, simulator complexity grows accordingly. That is why it is important not only to implement working software model of microprocessor, but also to have means to debug it and make it more accurate.

In this paper we described our approach to integration of memory subsystem model into application-level cycle-accurate simulator of Elbrus microprocessors and the tools we implemented and used to debug this simulator.

However, while the simulator described in this paper is reasonably performant and accurate (with mean running time error below 5%), there is always more work to be done.

Firstly, there are still some features and components of microprocessor that are not yet simulated with sufficient accuracy. Most notably, simulation of Array Access Unit (AAU) is still very simplified. Part of our future work is to properly implement more components. Also, one can always find bugs and inaccuracies that should be fixed.

For the memory subsystem timing model specifically, the authors plan to further validate the model and to reduce the error rate via fixing of simulator inaccuracies and adjustment of performance counters.

It is known that behavior of OS and effects of I/O can have significant effect on performance [16][17]. Such details are not properly captured by application-level simulators, unlike system-level simulators. Concrete examples of this can be seen in our evaluation. It is interesting to

explore ways to take into account such details (for example, stalls caused by DTLB, TLU and other components), while staying within AL simulation.

Secondly, while performance of simulator is reasonable enough and stays within our original requirements, complete simulation of many applications can take days of real time. It is important to improve this situation.

One obvious way to address this is to reduce inefficiencies of current implementation. The main advantage of this approach is that it does not negatively affect accuracy of simulation. The work on several optimizations of this sort is currently in progress.

Another way to speed up the simulation process is to identify and simplify the least impactful parts of the simulation. It is less obvious how to do it properly, but as accuracy and reliability of current simulator increases, it becomes easier to investigate and evaluate such options.

Thirdly, as it became evident during implementation of cycle-accurate simulator, proper tooling has significant impact on productivity. And it is quite reasonable to improve existing tools to further improve efficiency of debugging and developing of simulator. For example, it could be useful to add more types of events to execution statistics and to implement more verbose version of execution log.

Finally, the support of the upcoming version of the Elbrus ISA is yet to be done.

References / Список литературы

[1] Kutsevol V.N., Meshkov A.N., Chernykh S.V. The approaches to the performance optimization of multi-core «Elbrus» processors program models. Voprosy radioelektroniki, no. 3, 2017, pp. 57–61 (in Russian) / Куцевол В. Н., Мешков А. Н., Черных С. В. Методы оптимизации производительности программного моделирования многоядерных микропроцессоров с архитектурой «Эльбрус». Вопросы радиоэлектроники, no. 3, 2017 г., стр. 57–61.

[2] Poroshin P.A., Meshkov A.N. An exploration of approaches to instruction pipeline implementation for cycle-accurate simulators of «Elbrus» microprocessors. Proc. ISP RAS, 2019, vol. 31, no. 3, pp. 47-58. DOI: 10.15514/ISPRAS-2019-31(3)-4.

[3] Znamenskiy D.V., Kutsevol V.N. Development of a cycle-accurate simulator of the Elbrus processor core memory subsystem. Radio industry (Russia), vol. 29, no. 2, 2019, pp. 17-27 (in Russian), DOI: 10.21778/2413-9599-2019-29-2-17-27 / Знаменский Д.В., Куцевол В.Н. Разработка потактового симулятора подсистемы памяти процессорного ядра «Эльбрус». Радиопромышленность, том 29, no. 2, 2019 г., стр. 17–27.

[4] Henning J.L. SPEC CPU2006 benchmark descriptions. ACM SIGARCH Computer Architecture News, vol. 34, no. 4, 2006, pp. 1-17.

[5] Weidendorfer J. KCachegrind: Call graph viewer. Official Website. Available at: http://kcachegrind.github.io/html/Home.html, accessed April 10, 2020.

[6] Kozhin A.S., Neiman-zade M.I., Tikhorskiy V.V. Memory subsystem impact on the 8-core «Elbrus-8C» processor performance. Voprosy radioelektroniki, no. 3, 2017, pp. 13–21 (In Russian) / Кожин А.С., Нейман-заде М.И., Тихорский В. В. Влияние подсистемы памяти восьмиядерного микропроцессора «Эльбрус-8С» на его производительность. Вопросы радиоэлектроники, no. 3, 2017 г., стр. 13–21.

[7] Ermolitckii A.V., Neiman-Zade M.I., Chetverina O.A., Markin A.L., Volkonskii V.Y. Aggressive Inlining for VLIW. Proc. ISP RAS, 2015, vol. 27, issue 6, pp. 189-198 (in Russian). DOI: 10.15514/ISPRAS-2015-27(6)-13 / Ермолицкий А.В., Нейман-заде М.И., Четверина О.А., Маркин А.Л., Волконский В.Ю. Агрессивная инлайн-подстановка функций для VLIW-архитектур. Труды ИСП РАН, том 27, вып. 6, 2015 г., стр. 189-198.

[8] Yourst M.T. PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator. In Proc. of the 2007 IEEE International Symposium on Performance Analysis of Systems and Software, 2007, pp. 23-34.

[9] Alves M.A.Z., Villavieja C., Diener M., Moreira F.B., Navaux P.O.A. SiNUCA: A Validated Micro-Architecture Simulator. In Proc. of the 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and

Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, 2015, pp. 605-610.

[10] Akram A., Sawalha L. A survey of computer architecture simulation techniques and tools. Ieee Access, vol. 7, 2019, pp. 78120-78145.

[11] Carlson T.E., Heirman W., Eyerman S., Hur I., Eeckhout L. An evaluation of high-level mechanistic core models. ACM Transactions on Architecture and Code Optimization (TACO), vol. 11, no. 3, 2014, pp. 1-25.

[12] Woo S.C., Ohara M., Torrie E., Singh J.P., Gupta A. The SPLASH-2 programs: Characterization and methodological considerations. ACM SIGARCH Computer Architecture News, vol. 23, no. 2, 1995, pp. 24-36.

[13] Guthaus M.R, Ringenberg J.S., Ernst D., Austin T.M., Mudge T., Brown R.B. MiBench: A free, commercially representative embedded benchmark suite. In Proc. of the fourth Annual IEEE International Workshop on Workload Characterization, 2001, pp. 3-14.

[14] Bienia C., Kumar S., Singh J.P., Li K. The PARSEC benchmark suite: Characterization and architectural implications. In Proc. of the 17th International Conference on Parallel Architectures and Compilation Techniques, 2008, pp. 72-81.

[15] Aslot V., Eigenmann R. Performance characteristics of the SPEC OMP2001 benchmarks. ACM SIGARCH Computer Architecture News, vol. 29, no. 5, p2001, pp. 31-40.

[16] Vandierendonck H., De Bosschere K. On the Impact of OS and Linker Effects on Level-2 Cache Performance. In Proc. of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation, 2006, pp. 87-95.

[17] Cain H.W., Lepak K.M., Schwartz B.A., Lipasti M.H. Precise and accurate processor simulation. In Proc. of the Workshop on Computer Architecture Evaluation using Commercial Workloads, HPCA, 2002. vol. 8.

Information about authors / Информация об авторах

Pavel Alexeevitch POROSHIN received his MS degree from Moscow Institute of Physics and Technology in 2017. He is currently working as software engineer at INEUM. His research interests include software simulation of computing systems and cycle-accurate simulation of microprocessors.

Павел Алексеевич ПОРОШИН получил степень магистра в 2017 году в Московском физико-техническом институте. В настоящее время работает в ИНЭУМ им. И.С. Брука в качестве инженера-программиста. В область его научных интересов входят программное моделирование вычислительных систем и потактово-точное моделирование микропроцессоров.

Dmitry Valerievich ZNAMENSKIY – senior engineer at the MCST. His research interests include software modeling of computer systems and cycle-accurate simulation of microprocessors.

Дмитрий Валерьевич ЗНАМЕНСКИЙ – старший инженер в МЦСТ. В область его научных интересов входят программное моделирование вычислительных систем и потактово-точное моделирование микропроцессоров.

Alexey Nikolaevitch MESHKOV received his PhD degree at INEUM in 2013. He is currently working as a chief of department at MCST. His research interests include software modelling and verification of computer systems.

Алексей Николаевич МЕШКОВ получил степень кандидата технических наук в ИНЭУМ им. И.С. Брука в 2013 году. В настоящее время является начальником отдела в АО «МЦСТ». Область научных интересов включает программное моделирование и верификацию компьютерных систем.

DOI: 10.15514/ISPRAS-2020-32(2)-7

Исследование технологии RISC-V

[1,2] *В.А. Фролов, ORCID: 0000-0001-8829-9884 <vova@frolov.pp.ru>*
[1] *В.А. Галактионов, ORCID: 0000-0001-6460-7539 <vlgal@gin.keldysh.ru>*
[2] *В.В. Санжаров, ORCID: 0000-0001-6455-6444 <vadim.sanzharov@graphics.cs.msu.ru>*
[1] *Институт прикладной математики им. М.В. Келдыша РАН,*
125047, Москва, Миусская пл., д.4
[2] *Московский государственный университет им. М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1

Аннотация. Система команд – это стержень, вокруг которого строится весь остальной процессор. Ошибки или негибкость в решениях, однажды заложенные в систему команд, остаются с этим поколением процессоров навсегда. Поэтому одна из ключевых причин, по которой рост производительности современных CPU замедлился, заключается в том, что исходный код процессоров «испортился» в прямом и переносном смысле этого слова: процессоры внутри становятся сложными, из-за чего их дальнейшее развитие затрудняется. Разработка современных ЭВМ (CPU, GPU или специализированных систем) – это крайне дорогостоящий процесс, состоящий из большого количества затратных статей. Поэтому вопрос цены, или, скорее, целесообразности разработки процессора является ключевым. В данной работе мы провели исследование существующих популярных систем команд процессора и сделали выводы о перспективности в настоящее время направления RISC-V и других открытых систем команд CPU. Мы постарались ответить на следующие вопросы: почему система команд процессора – это действительно важно? Почему именно RISC-V, чем он лучше остальных? Какие возможности RISC-V открывает для российских разработчиков и какие у него есть аналоги?

Ключевые слова: RISC-V; система команд

Для цитирования: Фролов В.А., Галактионов В.А., Санжаров В.В. Исследование технологии RISC-V. Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 81-98. DOI: 10.15514/ISPRAS–2020–32(2)–7

Investigation of the RISC-V

[1,2] *V.A. Frolov, ORCID: 0000-0001-8829-9884 <vova@frolov.pp.ru>*
[1] *V.A. Galaktionov, ORCID: 0000-0001-6460-7539 <vlgal@gin.keldysh.ru>*
[2] *V.V. Sanzharov, ORCID: 0000-0001-6455-6444 <vadim.sanzharov@graphics.cs.msu.ru>*
[1] *Keldysh Institute of Applied Mathematics RAS,*
Miusskaya sq., 4, Moscow, 125047, Russia
[2] *Lomonosov Moscow State University,*
GSP-1, Leninskiye Gory, Moscow, 119991, Russia

Abstract. An Instruction Set Architecture (ISA) is the core around which the rest of the CPU is built. Errors or inflexibility in decisions once embedded in a system of instructions remain with this generation of processors forever. Therefore, one of the key reasons why the performance growth of modern CPUs has slowed down is that the source code of the processors “got corrupted” in literal and figurative sense of the word: the processors inside become complex which makes their further development difficult. The development of modern computers (CPU, GPU or specialized ones) is in any case an extremely expensive process consisting of a large number of costly articles. Therefore, the issue of cost of developing a processor

is the corestone. In this work we conducted a study of existing popular processor command systems and made conclusions about the prospects of the RISC-V and other open source instruction set architectures. We tried to answer the following questions: why the processor instruction set architecture is really important? Why RISC-V, why is it better than the others? Which opportunities does RISC-V open for developers around the world and what analogues does it have?

Keywords: RISC-V, Instruction Set Architecture

For citation: Frolov V.A., Galaktionov V.A., Sanzharov V.V. Investigation of the RISC-V. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 81-98 (in Russian). DOI: 10.15514/ISPRAS–2020–32(2)–7

1. Введение

Одна из наших основных целей как разработчиков – уметь писать высокоэффективные приложения. Проблема в том, что в современном мире это не так просто сделать. Среднестатистический код на C++ использует едва ли десятую долю производительности современных CPU [1]. Такая ситуация во многом сложилась от отсутствия прозрачного интерфейса между программистом и аппаратурой.

- Высокоуровневое алгоритмическое описание на C++ (или другом языке) транслируется в ассемблер с применением самых разных техник оптимизации вплоть до автоматической векторизации. На этом этапе компилятор активно использует машинно-зависимые оптимизации (то есть оптимизирует код для конкретного семейства процессоров).
- Ассемблер в действительности не позволяет получить представление о том, насколько данный код оптимизирован, даже несмотря на то, что он напрямую транслируется в машинный код. Вот здесь есть фундаментальная проблема.

Дело в том, что помимо архитектурного уровня, то есть непосредственно системы команд, видимой компилятору и программисту, существует ещё микроархитектурный – то, как именно эти команды выполняются внутри процессора. Таким образом, система команд – это не что иное, как интерфейс, а конкретное поколение процессоров (например, AMD Zen [2] или Intel Cascade Lake) – это реализация интерфейса. Когда интерфейс устаревает (перестаёт удовлетворять актуальным требованиям по какой-либо причине), программист его меняет. Однако существуют ситуации, когда интерфейс нельзя изменить – если его старой версией пользуются другие разработчики. Для программных систем это не является большой проблемой, поскольку старый интерфейс, как правило, может быть реализован через новый и, таким образом, можно двигаться вперёд, сохраняя поддержку для пользователей старого интерфейса какое-то время. Однако для аппаратуры это не так.

Проблема #1: Любая лишняя функциональность в системе команд будет тратить ресурсы кристалла (транзисторы, частоту, тепловыделение) и увеличивать сложность решения в целом. Дополнительные стадии конвейера, нацеленные, например, на сохранение частоты будут причиной роста латентности (непосредственное время выполнения) инструкций, что может повлиять на эффективность остальной системы. Поскольку обратная совместимость для CPU важна, с каждым новым поколением процессора, расширяющим базовый интерфейс, проблемы растут как снежный ком. Мы рассмотрим недостатки существующих систем команд в обзоре. Однако прежде чем мы к ним перейдём, необходимо упомянуть ещё несколько проблем.

Проблема #2: **Отсутствие кроссплатформенности.** Высокопроизводительные программные компоненты (библиотеки) на сегодняшний день очень сильно зависят от конкретной аппаратуры. Производители процессоров намеренно выпускают “открытый” софт, напичканный как можно большим числом своих инструкций. Например, Intel предоставляет бесплатную библиотеку трассировки лучей Embree (и многие другие бесплатные библиотеки), сверху до низу прошитую аппаратно-зависимыми «инстринсиками». Излишне упоминать о стоимости процессоров Xeon, под которые эта

библиотека оптимизируется. Перенести подобную библиотеку на процессор с другой системой команд – задача, эквивалентная по стоимости написанию своего аналога с нуля. Хуже всего, что перенос кода «в лоб» приведёт к низкой производительности, и, следовательно, не будет иметь смысла.

Приведём простой пример. В x64 есть инструкция shuffle, которая позволяет произвольным образом перераспределить содержимое векторного регистра. С помощью такой инструкции, например, можно реализовать векторное произведение для двух трёхмерных векторов, хранящихся в регистрах:

```
__m128  shuffle_yzxw(__m128 a_src)
{ return _mm_shuffle_ps(a_src, a_src, _MM_SHUFFLE(3, 0, 2, 1)); }

__m128 cross(const __m128 a, const __m128 b)
 {
    const __m128 a_yzx = shuffle_yzxw(a);
    const __m128 b_yzx = shuffle_yzxw(b);
    const __m128 c =
           _mm_sub_ps(_mm_mul_ps(a, b_yzx), _mm_mul_ps(a_yzx, b));
    return shuffle_yzxw(c);
 }
```

Однако оказывается, что эффективно реализовать на ARM аналог shuffle именно с таким порядком компонент (yzxw), как и в общем случае, невозможно. Но это не значит, тем не менее, что код, использующий векторные произведения, не может быть эффективно реализован на ARM. Просто вычисления и данные изначально должны быть организованы по-другому (например, полноценная векторизация кода с обработкой по 4 элементам, которая в общем случае даже на x64 работает лучше). Конечно, опытный читатель может возразить, что, ограничиваясь лишь конструкциями языка программирования (например, C++ или Ada), он может избежать проблем с переносимостью. Однако, даже если не брать в расчёт аппаратно-зависимые библиотеки, сохраняются нюансы (см. ниже).

Проблема #3: Компилятор. Разработчики процессоров (например, AMD) активно вносят свой вклад, особенно в gcc [2] и ядро Linux [3], потому что они в этом непосредственно заинтересованы. Компилятор и ОС – это такой же зависимый от производителя аппаратуры софт. Нельзя ожидать, что один и тот же код оптимизируется под разные процессоры одинаково хорошо, даже если они имеют практически идентичную функциональность (хотя LLVM и амортизирует эту проблему, но лишь отчасти).

Совершенно отдельный разговор – **безопасность системы** (***проблема #4***). Здесь всё очень плохо, потому что вам её, попросту говоря, никто не гарантирует (см. уязвимости «Spectre» и «Meltdown»). Хотите быть защищены – делайте целиком свой компьютер плюс ОС, компилятор и драйверы с нуля.

И, наконец, (***проблема #5***) – **эффективное взаимодействие потоков** в многопоточной программе – вопрос, не решённый в большинстве систем команд на должном уровне. Сейчас это особенно важно, так как процессорных ядер становится всё больше.

Таким образом, производители процессоров тянут одеяло на себя, зачастую намеренно добавляя «удобные» инструкции в свои процессоры. Разработчики программных систем начинают привыкать к этому «удобству», не понимая, к чему это приведёт в дальнейшем при переносе кода на процессор с другой системой команд. Мы не хотим сказать, что конкретно инструкция shuffle – это плохая идея. Мы хотим сказать, что этот вопрос не должен решаться одной компанией, если кроссплатформенность, обратная совместимость и безопасность для нас важны. Должен быть устойчивый стандарт, на который вы, как разработчик, можете опираться, чтобы ваша программа в будущем могла бы быть

перенесена на новые, лучшие, более быстрые, энергоэффективные или безопасные вычислительные системы без существенных потерь в производительности.

Замечание. Интересно упомянуть в этой связи графические процессоры. Поскольку обратная совместимость для них никогда не являлась целью, GPU прогрессировали существенно быстрее и достигли известных успехов. Нельзя сказать, что там всё просто, но проблемы переносимости ПО для них решаются на уровне графических и вычислительных API, что само по себе намного более гибко, чем система команд. В настоящее время производители GPU смогли договориться о едином открытом стандарте, называемом Vulkan, который поддерживают почти все современные десктопные и мобильные GPU. Как и RISC-V, Vulkan – хороший и грамотно спроектированный стандарт. Мы надеемся, что расскажем о том, почему это так в нашей следующей статье. А пока мы можем резюмировать, что, как бы странно это ни звучало, в современном мире программные системы и алгоритмы, интенсивно использующие GPU, в целом гораздо более кроссплатформенны, чем их CPU аналоги.

2. Обзор существующих систем команд

Прежде всего, необходимо разобраться в накопленном мировом опыте.

2.1 X86/X64

Intel и AMD добились успехов в распространении процессоров этой архитектуры благодаря обратной совместимости, агрессивным оптимизациям и лидирующим технологиям производства [4]. Дизайн набора инструкций – не их сильная сторона. Об этом говорит хотя бы то, что внутри этих процессоров x86/x64 инструкции давно уже транслируются в некоторое более простое, RISC-подобное представление. Всё это очень напоминает какую-нибудь старую программу с legacy функциональностью, которая до сих пор жива только потому, что все к ней уже привыкли. Но если отбросить совместимость и посмотреть объективно, в x86/x64 «просто нет смысла» ([4], с.12). А ведь это было сказано ещё в далёком 1994 г. [5]. Основная проблема x86/x64 – это исключительно раздутая (больше 2000) и плохо структурированная система команд. Многие команды уже давно никем не используются и не поддерживаются, но продолжают оставаться в системе команд в силу обратной совместимости. Вот несколько основных проблем.

Кодирование команд в x86/x64. В x86/x64 команды могут занимать разное число байт: от 1 до 15. При этом более короткие операции со временем стали использоваться реже. Изначальная идея была в том, чтобы кодировать наиболее частые операции меньшим числом байт (см. коды Хаффмана). Но со временем она повернулась обратной стороной. Например, целых 6 8-битных инструкций должны обрабатывать числа в десятеричном представлении, что уже давно не поддерживается в gcc ([4], с.12) и не реализуется современными процессорами, но по-прежнему занимает место в системе кодирования команд. Аналогично можно сказать и про весь сопроцессор x87, который хоть и используется в старых 32 битных программах, в 2020 году является атавизмом.

Регистры. В x86/x64 регистров очень много (рис. 1). Но в то же время, как ни странно, их очень мало. Много – из-за обратной совместимости x64 с x86 и его сегментными регистрами, x87 стеком FPU и прочими уже давно никем не используемыми вещами. Мало, потому что полезных регистров лишь 16 (а в x86 их всего 8). При этом, новые регистры всё время нарастают поверх старых (рис. 1), потому что 32 битные процессоры должны были уметь выполнять существующий бинарный 16 битный код, а 64 разрядные – 32 битный.

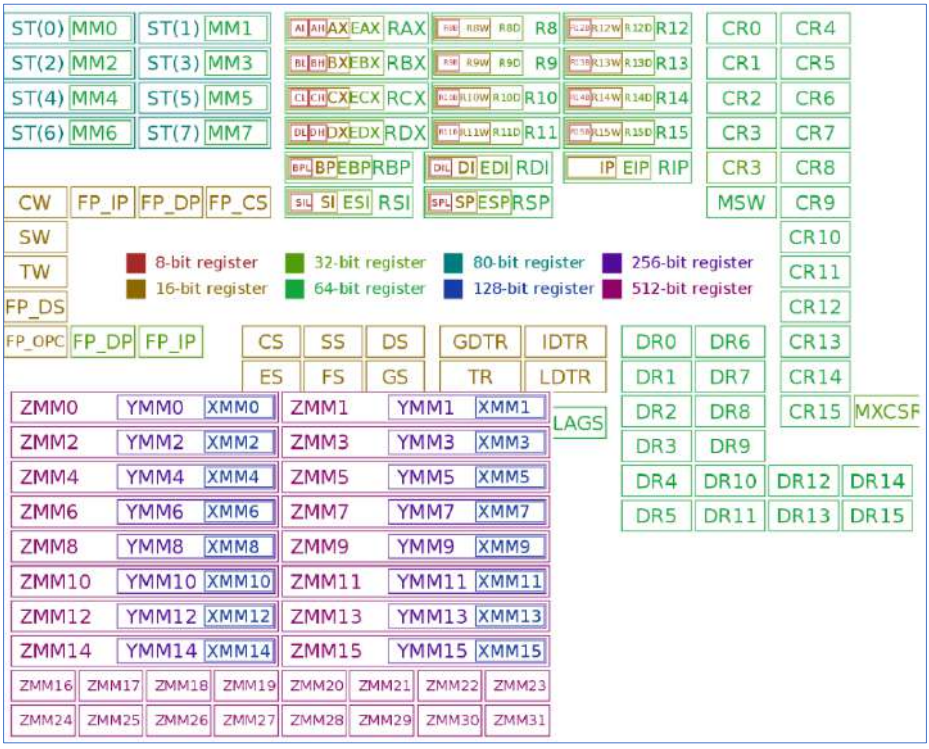

Рис. 1. Регистры x64
Fig. 1. x64 registers

Может показаться, что для современных чипов это число не так велико, но это совершенно не так: (1) Регистровая память в современных CPU может иметь большое количество портов и в действительности быть очень дорогой; (2) механизм переименования регистров, безусловно применяемый во всех современных CPU с внеочередным выполнением команд, использует в N раз больше физических регистров, чем в системе команд доступно логических (где N обычно от 2 до 4). Такое большое число неиспользуемых регистров граничит с преступлением. Но это ещё не всё и современный x64 обладает рядом других проблем.

- Двухадресные инструкции. Такие инструкции всегда перезаписывают один из операндов, что препятствует нормальным оптимизациям и вынуждает компилятор вставлять дополнительные операции «mov». Эта проблема даже породила своеобразный мем в программистской среде: «i like to mov it mov it».
- Некоторые предикатные инструкции призваны повысить производительность. Тем не менее, зачастую эта цель в x64 не достигается, т. к. их семантика плохо продумана. Например, в x64 есть команда условной загрузки. Однако, если для безусловной загрузки исключение при обращении по некорректному адресу строго обязательно, то для условной загрузки это не специфицируется. Из-за этого компилятор очень редко может использовать эту инструкцию для выполнения оптимизации, когда необходимо гарантировать корректность выполнения кода. Кстати говоря, именно эти инструкции активно используются в широко известных уязвимостях Spectre и Meltdown (которые, впрочем, затрагивают большинство современных архитектур, использующих спекулятивное исполнение команд). После исправлений критических уязвимостей в драйверах ОС производительность некоторых приложений наоборот упала на 30%.

- Некоторые регистры общего назначения в действительности такими не являются. Например, результат деления всегда будет находиться в паре DX/AX, а результат сдвига всегда поступает через DX. ESI/EDI также имеют специальную семантику. В целом такой шаблон проектирования приводит к неэффективному перемещению данных между регистрами и стеком [4].

2.2 ARMv7

На сегодняшний день x86/x64 аккумулирует в себе существенно больше недостатков, чем все другие системы. Неудивительно, что в области встроенных систем, где важна энергоэффективность, x86/x64 проиграла архитектуре ARM. Если сравнивать ARM и x86/x64, то ARM безусловно выглядит лучше. Однако и здесь проблем достаточно.

- Отсутствие поддержки 64-битного адресного пространства в ARMv7.
- ARMv7 – это, на самом деле не одна, а целых 3 системы команд: обычный режим и два сжатых: Thumb и Thumb 2. Из-за этого, в итоге, декодеры команд должны понимать три системы команд, что увеличивает энергопотребление, задержку и стоимость проектирования.
- ARMv7 по факту не является классическим RISC набором. Например, программный счётчик – это один из адресуемых регистров. А это означает, что почти любая инструкция может изменить поток управления (т. е. выполнить переход). Хуже всего то, что последний значащий бит счетчика программы отражает, какой именно набор команд в данный момент выполняется (ARM или Thumb), то есть обычная инструкция сложения может изменить то, какой именно набор команд в данный момент выполняется на процессоре!
- Использование кодов условий для переходов и предикации по факту лишь усложняет высокопроизводительные реализации с внеочередным выполнением команд, хотя может быть плюсом для процессоров с очередным выполнением или простым внеочередным выполнением на основе scoreboard.
- Совершенно отдельная история – комплексные команды для вызова функций вроде «LDMIAEQ SP, R4-R7, PC», которая «выполняет 6 загрузок, увеличивает счётчик программ и записывает 7 регистров в стек, включая адрес возврата … и, кроме того, это условная команда, то есть на самом деле это ещё одновременно и условный переход» [4].
- Наконец, ARMv7 – это очень большая система команд, включающая в себя более 600 инструкций даже без плавающей точки и SIMD.

Конечно, вопрос о вреде или пользе комплексных команд, как и вопрос о раздутости самой системы команд, не очевиден: много команд – это хорошо или плохо? Сложные инструкции – это хорошо или плохо? Попробуем объяснить нашу позицию, используя аналогию. Когда вы разрабатываете программный интерфейс, он должен максимально простым и очевидным способом выражать то, что происходит в вашей системе и отражать те цели, которые вы преследуете. Он не должен быть слишком высокоуровневым или, наоборот, слишком низкоуровневым для ваших задач. Он не должен содержать лишних функций, так как это усложнит поддержку в будущем. Система команд процессора – это квинтэссенция интерфейса между программой и аппаратурой, – и эта квинтэссенция должна содержать только то, что действительно будет реализовано на аппаратном уровне. Поэтому для ответа на этот вопрос следует обратить внимание на существующий опыт и для каждого подобного случая принять взвешенное решение. Если среди целей мы видим переносимость, простоту, обратную совместимость и безопасность, то, вероятно, от комплексных команд лучше отказаться.

2.3 ARMv8

В 2011 году, через год после начала проекта RISC-V, ARM анонсировала полностью переработанную систему команд ARMv8 с 64-битными адресами и расширенным набором целочисленных регистров. В новой архитектуре инженеры ARM убрали функции ARMv7, которые усложняли реализацию: счетчик команд больше не является частью набора целочисленных регистров; предикатных инструкций больше нет; комплексные инструкции с множественной загрузкой и сохранением были удалены, а кодирование команд в целом упрощено. Однако не все проблемы были решены, и, кроме того, добавились новые.

- Условные коды по-прежнему используются в операциях перемещения cmov, что создаёт определённые проблемы для переименования регистров: если условие не выполняется, команда все равно должна скопировать старое значение в новый физический регистр. Это фактически делает условное перемещение единственной инструкцией, читающей три исходных операнда вместо двух.
- ARMv8 не является модульной. Например, SIMD инструкции, предполагающие наличие 32 «жирных» векторных регистров строго обязательны к реализации. Это не подходит для многих встроенных решений, где важны низкие себестоимость и энергопотребление.

В целом ARMv8 является очень большой и напичканной самой разной функциональностью коммерческой системой команд: более 1000 инструкций в 53 форматах, описанные примерно на 6000 страниц документации. При этом, некоторые важные вещи всё же упущены: например, отсутствует какой-либо аналог Thumb (что важно для многих встроенных систем в силу ограниченной памяти для кода), нет объединённых инструкций сравнения + перехода. Кроме того, такая функциональность как размещение констант непосредственно в коде инструкции (так называемый immediate формат инструкции) в обеих версиях ARM ограничена: нет возможности загрузить произвольную 32 битную константу непосредственно из кода программы за 1 или, хотя бы, за 2 инструкции.

2.4 MIPS

Основная проблема MIPS – недостаточная гибкость системы команд. Если x64 и ARM зачастую содержат слишком высокоуровневые инструкции, то в MIPS перекос идёт в другую сторону: команды опускаются до уровня микроархитектуры, предполагая строго определённую реализацию в аппаратуре. Например, результат умножения и деления сохраняется в специальном внутреннем 64 битном регистре. То есть результат нужно получать отдельной командой даже если вам нужны только 32 младших бита.

```
mult  R2, R3
mfhi  R4
mflo  R5
```

По факту старшие 32 бита почти всегда отбрасываются (как вы думаете, какой тип будет иметь результат умножения двух 32 битных чисел в gcc с настройками по умолчанию?), а используется только младшая часть. Здесь мы видим задержку, искусственно введённую в систему команд (ещё одним примером такой задержки в MIPS является так называемая Branch Delay Slot – команда «nop», которая вставляется сразу за командой перехода). Проблему разной длины конвейера для разных команд можно решить, реализовав простой вариант внеочередного выполнения команд (scoreboard), но оставаясь при этом в рамках полного детерминизма и простоты. С другой стороны, наличие этого дополнительного внутреннего состояния усложняет суперскалярную реализацию с внеочередным выполнением команд ([4], стр. 4).

С плавающей точкой в MIPS также не всё хорошо. Обратите внимание, что в MIPS ассемблере инструкции, работающие с двойной точностью, используют только чётные

номера регистров. Так происходит, поскольку числа с двойной точностью занимают 2 смежных регистра – например, пару (f2, f3) при обращении к f2.

```
add.s    $f1, $f0, $f1      # single precision add
sub.s    $f0, $f0, $f2      # single precision sub
add.d    $f2, $f4, $f6      # double precision add
sub.d    $f2, $f4, $f6      # double precision sub
```

Такое решение имеет свои плюсы. Поскольку двойная и одинарная точность обычно не смешиваются в C++ коде, можно более эффективно использовать один и тот же набор физических регистров: либо для того, либо для того. Это контрастирует, например, с x64 и ARMv8, в которых для двойной точности используется половина 128 битного регистра, а для одинарной – одна четверть. Кроме того, преобразования из одинарной точности в двойную могут быть реализованы относительно безболезненно (правда для x64 и ARMv8 это тоже верно). Однако есть и минус. Использование смежных пар регистров усложняет суперскалярную реализацию с внеочередным выполнение команд при реализации переименования регистров: логически смежные после переименования регистры перестают быть таковыми в физическом регистровом файле ([4], стр.6, п.3).

2.5 SPARC

Отличительная черта SPARC это регистровые окна. На первый взгляд идея не такая уж плохая. Вы расширяете фактическое количество регистров при помощи приёма «база + смещение». Пишете R1, а фактически подразумеваете SP+R1. Удобно. Кусочек стека сохраняется на регистрах. При вызове функций не нужно ничего никуда перемещать, да и компилятор становится, вроде как, немного проще.

Однако регистровые окна дороги в плане реализации. Огромное количество регистров, находящихся «в стеке», по факту не используются (напомним, что регистры всё-таки дороже чем L1 кэш-память из-за необходимости многопортовой памяти; например, в Эльбрусе, который основан на SPARC, регистровый файл содержит 20 портов, что на наш взгляд очень много ([6], с.126). Кроме того, когда регистровая память всё-таки заканчивается, наступает апокалипсис в операционной системе ([4], с.6).

Справедливости ради следует сказать, что можно было бы реализовать стек регистров через кольцевой буфер и выполнять параллельно откачку/подкачку неиспользуемой части регистров в память. Но для этого нужно делать ещё больше портов в регистровой памяти! Что-то похожее наблюдается в уже упомянутом Эльбрусе ([6], с.139), но в методичке [6] эта тема на наш взгляд раскрыта недостаточно полно. В итоге современные исследователи сходятся во мнении: если у вас есть лишние транзисторы, лучше просто сделать больше регистров и предоставить компилятору развлекаться со встраиванием функций. Не стоит перекладывать эту задачу на уровень аппаратуры.

Однако есть ещё одна ложка дёгтя. В SPARC существует отдельный набор регистров для плавающей точки (организован аналогично MIPS). Передача между целочисленными регистрами и регистрами с плавающей точкой осуществляется только через память. А ещё есть 8 глобальных регистров. Если процедура использует арифметику с плавающей точкой или меняет глобальные регистры, их нужно сохранять в стеке. Но в каком именно стеке? То есть у нас их теперь два, один в памяти для плавающей точки и глобальных регистров и один на регистрах для целых чисел. Если всё-таки стек только один, то мы должны сначала сохранить данные во временные ячейки памяти только для того, чтобы потом поместить их в наш регистровый стек, где они, кстати говоря, никак не будут использоваться. В результате идея регистровых окон для плавающей точки в SPARC не работает.

2.6 Power

Система команд Power появилась в результате эволюции и развития систем команд, разрабатывавшихся в первую очередь компанией IBM. Ход развития выглядит так:

POWER −> PowerPC −> Power ISA v2.x −> Power ISA v3.x.

С одной стороны, Power – это довольно сложная система команд, которая разрабатывалась главным образом компанией IBM, которая не славится открытостью. С другой стороны, с недавних пор эта архитектура стала полностью открытой на весьма привлекательных условиях [7]. Хотя систему команд Power нельзя назвать простой – чуть меньше 1000 инструкций в 25 форматах (включая векторные инструкции), все же она не столь сложна как ARM. Кроме того, особенностью Power является модульность – каждый регистр относится к функциональному классу, а большинство инструкций внутри класса используют только принадлежащие к этому классу регистры. Только небольшое количество инструкций передают данные между разными функциональными классами. К функциональным классам относятся: условный, с фиксированной точкой, с плавающей точкой и векторный. Таким образом, конкретная реализация может не поддерживать, например, операции с плавающей точкой (как некоторые встроенные решения на архитектуре Power). И «отключить» поддержку оказывается относительно просто. Также такое разделение функциональности в процессоре дает информацию о зависимости между регистрами компилятору. Однако у такого подхода неизбежно появляются и недостатки – взаимодействие между разными функциональными классами может приводить к задержкам, зависимым от реализации. Забегая вперёд, скажем, что этот подход очень напоминает RISC-V и, возможно, именно поэтому разработчики RISC-V ***умалчивают о системе команд Power*** в работе [4].

Отдельно следует упомянуть условные регистры. Также как в ARM и SPARC, в Power есть специальные регистры, в которых выставляются 4 бита состояния после выполнения операций сравнения. Но в Power присутствуют 8 их копий. Каждая из этих копий может быть результатом инструкции сравнения и каждая копия может быть источником ветвления. Такая избыточность применяется, чтобы инструкции могли использовать разные условные состояния без конфликта (когда в разных ветвях кода получаются разные состояния для последующих ветвлений). Кроме того, предоставляется возможность совершать логические операции между этими 4-х битными регистрами, что позволяет одной ветви проверять более сложные условия.

Ещё одна особенность Power – два специальных регистра Link и Count. Регистр Link используется вместо регистра общего назначения (как во многих других архитектурах) для хранения адреса возврата при вызове процедуры. Регистр Count используется как счетчик в циклах с фиксированным числом итераций. Используя этот специальный регистр, аппаратное обеспечение ветвления может быстро определить вероятное ветвление, так как значение регистра известно в начале цикла выполнения. Кроме того, оба регистра могут содержать адрес условной ветви и имеются специальные инструкции, позволяющие получить этот адрес. Подобное решение с одной стороны дает определенные преимущества в скорости при обработке ветвлений, а с другой вносит дополнительную сложность в реализацию.

В системе команд Power есть ряд и других специфичных инструкций – множественной загрузки и сохранения (до 32 регистров), генерации случайного числа в регистре, набор из 48 инструкций для поддержки десятичной плавающей точки и др. В плане векторного расширения можно отметить поддержку 128-битных чисел с фиксированной точкой, а также расширения, добавляющего векторно-скалярные инструкции. Таким образом, Power обладает рядом довольно специфичных особенностей, которые могут быть как плюсом (производительность), так и минусом (сложность). Эта специфичность нашла свое

отражение и в существующих применениях и реализациях этой системы команд - суперкомпьютеры (первые две строчки в рейтинге TOP500 занимают компьютеры на базе Power9) и микроконтроллеры (главным образом в автомобильной и авиационной отраслях). Среди других примеров реализации системы команд Power – одна из компонент процессора Cell Broadband Engine игровой консоли Playstation 3, а также компьютеры Apple Power Mac, выпускавшиеся до 2006 года.

Разнообразие применений Power привело к тому, что одна из основных проблем Power заключается не в системе команд как таковой, а в экосистеме, которая создается пользователями – разработчиками. Узкоспециализированные применения не дают сформироваться широкому сообществу разработчиков, способных разрабатывать эффективные приложения под эту систему команд и способствующих её поддержке и развитию.

Однако в последние годы эта ситуация начала меняться. Появились доступные обычному пользователю персональные компьютеры, использующие процессор Power9 [8], отличающиеся открытостью «прошивок» (firmware) загрузчика и других «сопутствующих» компонентов. Стала открытой сама система команд, появились реализации в FPGA с открытым исходным кодом [9]. А проект по разработке открытого GPU, исходно называвшийся Libre-RISCV [10], и, как следует из названия, ориентировавшийся на RISC-V, переключился на архитектуру Power (причины, правда, скорее политические [11]). Развивается проект по созданию открытого ноутбука на базе PowerPC [12]. Но все же пока что все реализации архитектуры Power в кремнии происходили при непосредственном участии IBM. Получится ли у Power, ставшей открытой, догнать RISC-V или время все же упущено, и IBM, однажды потеряв рынок персональных компьютеров, вновь повторила свою ошибку – будущее покажет. На наш взгляд Power – достойный конкурент RISC-V.

3. Исследование системы команд RISC-V

Система команд RISC-V разрабатывается в университете Berkley. Их основная концепция звучит следующим образом: RISC-V должна стать единой системой команд для всех типов ЭВМ от микроконтроллеров до высокопроизводительных. В действительности, за этим перфекционистским лозунгом стоит ряд практических и очень конкретных целей.

1. **Повышение энергоэффективности и производительности**, уменьшение числа необходимых транзисторов, удешевление разработки и поддержки процессорных ядер за счёт грамотно созданного интерфейса: разработчики процессорных ядер в минимальном варианте обязаны реализовывать лишь относительно небольшой набор инструкций (по сравнению с остальными открытыми и закрытыми системами команд) для получения полноценно работающей системы. После чего они могут сосредоточиться на интересующих именно их областях.
2. **Модульность и расширяемость.** Минимальных наборов команд на сегодняшний день существует достаточно много и RISC-V далеко не самый минимальный из них. Основная проблема существующих систем команд в том, что небольшой набор команд не универсален. Он бывает ориентирован, например, для высокопроизводительных вычислений или областей, требующих специфической аппаратной функциональности (например, многопоточность, графика, криптографические системы). RISC-V призван решить эту проблему благодаря модульной структуре системы команд. Разработчики ЭВМ могут реализовывать существующие расширения (например, RVF32 для работы с плавающей точкой или RVA32 для атомарных операций), либо добавлять свои собственные – система команд спроектирована с прицелом на расширение. То есть предполагается, что вы, как разработчик конкретной системы, будете её расширять. Это в корне отличает RISC-V, например, от ARM, где не только расширяться уже очень

проблематично, но и сама компания ARM прямо запрещает подобные расширения по условиям лицензии.

3. **Удешевление разработки** различных электронных систем за счёт создания единой открытой экосистемы: компиляторы, операционные системы, драйверы, периферия. Наличие открытых репозиториев, в которых многое уже реализовано и за счёт многолетней поддержки сообщества доведено до высокого качества, не нуждается в дополнительных комментариях.
4. **Обратная совместимость** программного обеспечения и работа на будущее – программы, разработанные для старых процессоров (вернее, процессоров с поддержкой меньшим количеством расширений), должны работать на новых (вернее, процессорах той же системы команд, но с большим количеством реализованных расширений) без перекомпиляции. Плюс совместимость библиотек на бинарном уровне. То есть, скомпилированная статическая библиотека для одной ОС и одного процессора может быть использована как есть на другой ОС с другим процессором благодаря совместимости на уровне компилятора и собственно процессора. Здесь, конечно, следует сделать одну оговорку. Если в программе вы используете некоторое существующее расширение (например, плавающую точку) или, тем более, какое-то своё специальное расширение, то как есть эта библиотека на другой системе, возможно, не заработает. Тем не менее, благодаря наличию спецификации расширений ситуация поправима! Компилятор может перекомпилировать вашу библиотеку, заменив, например, команды плавающей точки или команды любого другого расширения на вызовы соответствующих функций с программной реализацией. В случае зоопарка коммерческих систем команд вроде x64 или ARM это сделать существенно труднее.
5. **Безопасность #1 (Security).** Если в некотором RISC-V процессоре обнаружена уязвимость, вы можете легко заменить его на другой RISC-V процессор, пусть и не такой эффективный, но зато без закладок потенциального противника. При этом вы сможете запустить на нём всё существующее ПО с минимальными затратами усилий и времени.
6. **Безопасность #2 (Safety).** Программное обеспечение, от которого зависит жизнь людей, проходит специальную процедуру сертификации и иногда формальной верификации, где корректность программы доказывается математически. Стоимость этих процедур может многократно превышать затраты на непосредственную разработку ПО. Однако, даже если вы, например, доказали корректность вашего ядра операционной системы, процедура сертификации – это отдельная проблема и зачастую сменить аппаратуру при этом не представляется возможным. По этой причина система команд Power на сегодняшний день так прочно обосновалась в области гражданской авиации. Стандарт RISC-V призван удешевить процедуру сертификации поскольку система команд и софт по большей части остаются неизменными (Имеется ввиду, что какая-то часть вашей системы будет меняться так или иначе в силу требований того или иного заказчика. Важно, чтобы эта часть оставалась небольшой, тогда затраты на сертификацию будут низкими).

3.1 Базовый набор инструкций RISC-V

Система команд RISC-V имеет следующие отличительные черты:

1. **Отсутствие неявных внутренних состояний.** Результат любой операции (кроме команд перехода) всегда помещается в регистр общего назначения. То есть в RISC-V нет, например, флагов состояний, которые устанавливает инструкция cmp в x86. Вместо этого результат команды сравнения помещается в один из регистров общего назначения. Такой подход существенно упрощает суперскалярную реализацию с

внеочередным выполнением команд, но при этом по сравнению с флагами или другими состояниями он не вносит существенных накладных расходов для простых реализаций.

2. **Отсутствие предикатных инструкций.** Вернее, их нет в базовом наборе, но они есть в векторном расширении. Предикатные инструкции совершенно необходимы для VLIW процессоров (Эльбрус) и для векторной обработки данных. Однако, выигрыш в скалярном коде от них обычно не очень большой. Для повышения эффективности ветвлений разработчики RISC-V предлагают добавлять простую реализацию предсказателя ветвлений (т. н. Branch Target Buffer в виде небольшой таблицы адресов перехода), которая дополнительно позволяет избавиться и от Branch Delay Slot. С другой стороны, в отсутствие таких инструкций она упрощает реализацию суперскалярного процессора с внеочередным выполнением команд.
3. **Компактный базовый набор инструкций.** В базовом наборе (рис. 2) всего 11 базовых арифметических инструкций (большинство из них могут встречаться в 2-х формах, давая в сумме 21 инструкцию), 10 инструкций для обращения в память и 8 инструкций для переходов. Итого – 39 инструкций.
4. **Загрузка произвольной 32 битной константы** из памяти инструкций за 2 команды (liu + следующая команда в I-формате). При этом большая часть констант длиной в 12 бит или меньше загружаются из кода программы за 1 инструкцию (собственно I-формат).
5. **32 регистра общего назначения**, что в 2 раза больше чем в большинстве RISC аналогов.
6. **Наличие сжатого 16-битного представления** команды (RV32C) аналогично ARM Thumb для микроконтроллеров.
7. **Поддержка так называемой ослабленной модели памяти** (Relaxed Memory Model) в базовом наборе инструкций (рис. 3). Это не настоящие атомарные операции, для них есть специальное расширение RV32A. Ослабленная модель памяти – это более явное (чем принято традиционно) выражение синхронизации данных между потоками в многопоточной программе. В C++ для этого существует специальная часть стандартной библиотеки (см. std::memory_order). Дело в том, что передача данных между потоками – это в действительности очень непростая и зачастую дорогостоящая операция с учётом многоуровневого кэша в современных процессорах. Традиционно принятая сильная модель памяти, когда любое изменение в памяти, сделанное в одном потоке должно быть немедленно увидено в другом потоке, является одним из краеугольных камней, ограничивающих рост производительности при росте количества ядер. Ослабленная модель памяти помогает программисту более явно выразить его намерения и, например, не выполнять дорогостоящую операцию синхронизации данных через L2/L3 кэш при записи в ячейку памяти, если она не нужна программисту в данный момент.
8. **Наличие спаренных (fused)** операций сравнения + переход, уменьшающие размер программного кода.
9. Инструкции, **ускоряющие вызов функций** (jal), вызов виртуальных функций и выполнение операторов switch (jalr) также присутствуют.

Instruction	Format	Meaning
`add rd, rs1, rs2`	R	Add registers
`sub rd, rs1, rs2`	R	Subtract registers
`sll rd, rs1, rs2`	R	Shift left logical by register
`srl rd, rs1, rs2`	R	Shift right logical by register
`sra rd, rs1, rs2`	R	Shift right arithmetic by register
`and rd, rs1, rs2`	R	Bitwise AND with register
`or rd, rs1, rs2`	R	Bitwise OR with register
`xor rd, rs1, rs2`	R	Bitwise XOR with register
`slt rd, rs1, rs2`	R	Set if less than register, 2's complement
`sltu rd, rs1, rs2`	R	Set if less than register, unsigned
`addi rd, rs1, imm[11:0]`	I	Add immediate
`slli rd, rs1, shamt[4:0]`	I	Shift left logical by immediate
`srli rd, rs1, shamt[4:0]`	I	Shift right logical by immediate
`srai rd, rs1, shamt[4:0]`	I	Shift right arithmetic by immediate
`andi rd, rs1, imm[11:0]`	I	Bitwise AND with immediate
`ori rd, rs1, imm[11:0]`	I	Bitwise OR with immediate
`xori rd, rs1, imm[11:0]`	I	Bitwise XOR with immediate
`slti rd, rs1, imm[11:0]`	I	Set if less than immediate, 2's complement
`sltiu rd, rs1, imm[11:0]`	I	Set if less than immediate, unsigned
`lui rd, imm[31:12]`	U	Load upper immediate
`auipc rd, imm[31:12]`	U	Add upper immediate to pc

Рис. 2. Вычислительные инструкции RISC-V. Целочисленные инструкции из базового набора RISC-V. В действительности различных инструкций всего 11, поскольку большая часть инструкций просто встречается в 2-х форматах – R-формат (стандартный) и I-формат (когда второй операнд считывается прямо из самой инструкции). Следует сказать, что хранить константы в коде – это отличная идея, повышающая производительность. Поскольку инструкции так или иначе уже находятся в чипе, прочитать константу из памяти инструкции ничего не стоит

Fig. 2. Computational instructions of RISC-V. Integer instructions from the RISC-V core set. In reality, there are only 11 different instructions, since most of the instructions are just found in 2 formats – R-format (standard) and I-format (when the second operand is read directly from the instruction itself). It should be said that storing constants in code is a great idea that improves performance. Since instructions are already on the chip one way or another, reading a constant from the instruction memory is worthless

Instruction	Format	Meaning
`lb rd, imm[11:0](rs1)`	I	Load byte, signed
`lbu rd, imm[11:0](rs1)`	I	Load byte, unsigned
`lh rd, imm[11:0](rs1)`	I	Load half-word, signed
`lhu rd, imm[11:0](rs1)`	I	Load half-word, unsigned
`lw rd, imm[11:0](rs1)`	I	Load word
`sb rs2, imm[11:0](rs1)`	S	Store byte
`sh rs2, imm[11:0](rs1)`	S	Store half-word
`sw rs2, imm[11:0](rs1)`	S	Store word
`fence pred, succ`	I	Memory ordering fence
`fence.i`	I	Instruction memory ordering fence

Рис. 3. Инструкции RISC-V для работы с памятью, включая специальные инструкции fence для синхронизации данных в многопоточных программах

Fig. 3. RISC-V instructions for working with memory, including special fence instructions for synchronizing data in multi-threaded programs

Таким образом, исключительно компактный набор менее чем в 40 инструкций содержит внушительный объем полезной функциональности.

3.2 Плавающая точка в RISC-V

Расширение RV32F, добавляющее плавающую точку, добавляет 32 новых регистра. Сразу обращает на себя внимание наличие инструкций преобразования в целые числа и обратно и, кроме того, инструкции непосредственного перемещения данных из целочисленного регистра в регистр с плавающей точкой, что является проблемой для многих существующих систем команд, когда данные приходится перемещать через память. Для двойной точности в RV32D операций перемещения нет, поскольку регистры двойной точности занимают в 2 раза больше бит (тем не менее, они есть в RV64D), но операции преобразования плавающей точки в целочисленную и обратно в RV32D сохраняются. Ещё обращает на себя внимание наличие инструкций FMA (спаренная операция умножения и сложения/вычитания), существенно сокращающих размер кода и повышающих производительность.

3.3 Векторное расширение RISC-V

Система команд RISC-V содержит ещё много интересных расширений. Мы хотим обратить внимание читателя на расширение RV32V, предназначенное для реализации векторных операций. Оно имеет следующие ключевые особенности.

1. **Длина вектора не фиксируется** в наборе инструкций, а задаётся в программе специальной инструкцией. Это в корне отличается от существующих подходов и позволяет вам выполнять на процессорах с небольшой длиной вектора программы, скомпилированные под более широкий вектор. Пусть и с меньшей производительностью. В современных x64 процессорах, например, это не так. Если вы собрали вашу программу, например, с AVX512, то эта программа сможет выполняться только на дорогих моделях Xeon и последних процессорах 109*0X, содержащих широкие векторные регистры zmm.
2. **Предикатное выполнение**, то есть наличие масок во всех векторных операциях. Такой подход одинаково хорош как для обычных CPU программ, так и для реализации массивно-параллельных систем (OpenCL и аналоги).
3. **Векторные регистры** существуют отдельно от остальных регистров, а перемещение данных между скаляром и вектором также поддерживается аппаратно. Благодаря этому в RISC-V отсутствуют неожиданные нюансы в смешанном скалярно-векторном коде, как это есть, например, в x64 (и особенно в x86), где зачастую лучше явно использовать векторный тип __m128 вместо float или int, если вы хотите быть уверены в том, что данные в память из векторных регистров лишний раз не выгружаются.
4. Другие необходимые в векторном коде операции вроде загрузки данных из памяти с определённым шагом, **gather и scatter**, а также поддержка матриц.

В целом можно сказать, что векторное расширение спроектировано с учётом опыта многих существующих архитектур и, как и остальная часть системы команд, выполнено грамотно. Интересно в этой связи отметить такой проект как библиотеку enoki, которая уже сегодня пытается эмулировать похожую функциональность программно на x64 и ARM, а в gcc варьируемая длина вектора в настоящий момент реализована на уровне компилятора (правда в gcc она должна быть кратна 4) [13].

4. RISC-V сегодня

В настоящий момент в сообщество RISC-V входит большое число известных компаний (рис 4). Из наиболее интересных мы бы хотели упомянуть компанию Nvidia, использующую RISC-V и язык Ada для создания беспилотных автомобилей, и компанию Alibaba, которая представила летом 2019 г. первый IP-блок для искусственного интеллекта. Кроме того, RISC-V уже получает популярность на уровне крупных государственных структур: Индия объявила RISC-V национальным стандартом, US DARPA требует RISC-V в качестве обязательного компонента по ряду программ, в том числе – по всему направлению HW

security research, Israel Innovation Authority создает общую платформу GenPro на базе RISC-V, в Китае объявлена объемная программа гос. субсидирования решений на базе RISC-V (2018), а Европейский Союз обсуждает крупную программу высокопроизводительных вычислений на основе RISC-V. Кроме того, RISC-V становится основой учебных программ по Computer Science и Electronic Engeneering направлений во многих университетах. Наконец, RISC-V проник и в Россию [14].

Рис. 4. Некоторые из компаний, участвующих в разработке экосистемы RISC-V и использующих эту экосистему в своих решениях

Fig. 4. Some of the companies involved in the development of the RISC-V ecosystem and using this ecosystem in their decisions

Мы уверены, что можно найти ещё примеры, доказывающие зрелость стандарта. Важно не это, а то, что на сегодняшний момент RISC-V уже имеет сформированную экосистему:

- Open Source ПО: GCC, Linux, BSD, LLVM, QEMU, FreeRTOS, ZephyrOS, LiteOS, SylixOS;
- коммерческое ПО: Lauterbach, Segger, Micrium, ExpressLogic и др.;
- открытые процессорные ядра: Rocket, BOOM, RI5CY, Ariane, PicoRV32, Piccolo, SCR1 (Российское ядро от Syntacore), Hummingbird и др.;
- коммерческие процессорные ядра: Codasip, Cortus, C-Sky, Nuclei, SiFive, Syntacore (Российская) и др.;
- использование RISC-V внутри компании: Nvidia, Western Digital, Qualcom, CloudBear (Российская) и др.

4.1 Перспективы для отечественных разработчиков

Нельзя не упомянуть наиболее известные отечественные процессоры – Эльбрус от МЦСТ и немного менее известные NMC4 (нейроматрикс) от НТЦ Модуль, и «мультикор» от Элвис. Как Эльбрус, так и NMC4 являются процессорами VLIW со своими компиляторами и целиком своей инфраструктурой. С технической точки зрения, на наш взгляд, это очень хорошие решения, особенно когда необходимо добиваться высокой производительности. Например, современный Эльбрус является абсолютным рекордсменом по числу команд в такт (до 50) и на многих задачах побеждает последние новинки от Intel и AMD. Но эти решения дорогие и по большей части закрытые.

Проблема большинства отечественных компаний в том, что их решения не следуют открытым стандартам в достаточной степени, из-за чего даже внутри нашей страны их можно использовать лишь ограниченно. Пересекаясь с отечественными разработками в области ПО для гражданской авиации, мы встречаем нежелание разработчиков связываться с закрытой и специфической экосистемой Эльбруса, как и с любой другой закрытой и малопонятной системой в принципе. При этом необходимо помнить, что для авиации, как и для многих других систем необходимы процедуры сертификации, где закрытые решения недопустимы. Следует сказать и о том, что экспортные системы вооружения зачастую также поставляются с иностранной электроникой, потому что нашей (как и любой другой закрытой системе) заказчики не доверяют.

При всех её преимуществах, к сожалению, VLIW-технология (используемая Эльбрусом и NMC4) никак не состыкуется с идеологией RISC-V, поскольку противоречие заложено в корне: если RISC-V подход *предоставляет интерфейс,* разделяя архитектурный и микроархитектурный уровни, то VLIW-подход прямо противоположен: система команд формируется непосредственно под микроархитектурный уровень конкретного процессора с конкретным числом блоков определённого типа. То есть, например, если у двух разных VLIW-процессоров разное количество блоков умножения с плавающей точкой (и больше они ничем не отличаются), то компилятор генерирует для них разные программы. С другой стороны, это не значит, что для VLIW-подхода стандартизация и открытость невозможна в принципе. Например, это может быть OpenCL или Vulkan: если рассмотреть выполнение рабочей группы потоков (например, из 256 или 512 элементов) в массивно-параллельной модели как цикл, тогда такой цикл, очевидно, можно конвейеризовать программно. Поскольку в большинстве случаев потоки OpenCL работают над данными независимо, механизм программной конвейеризации циклов должен показать хорошие результаты.

4.2 Недостатки RISC-V

Несмотря на ряд очевидных преимуществ RISC-V по сравнению с аналогами, есть ряд нюансов:

1. Любая организация по стандартизации подобная консорциуму RISC-V (или, например, Khronos) имеет две стороны медали. Одна сторона – это открытость, совместимость и лёгкость вхождения в область для небольших компаний. Другая сторона состоит в ограничении гибкости – предложить своё расширение системы команд позволяется лишь очень крупным спонсорам и даже для них этот процесс не простой. Как мы уже упоминали, проект по разработке открытого GPU, называющийся Libre-RISCV (так и не сменивший название), переключился на архитектуру Power, ссылаясь на невозможность получить необходимую документацию от сообщества. Причины конфликта понятны: разработчики Libre-RISCV предлагали векторное расширение (названное ими Simple-V), которое, по-видимому, было не интересно сообществу RISC-V из-за готовности собственного векторного аналога.
2. Когда речь идёт о высокой производительности, практически во всех материалах по RISC-V видно очень сильное смещение в сторону суперскалярных ядер с внеочередным выполнением команд. С одной стороны, это ожидаемо, поскольку такой способ повышения производительности является логическим развитием идеологии RISC в принципе. С другой стороны, это выглядит как лоббирование определённой технологии с целью продвижения собственных разработок.
3. RISC-V всё-таки не является панацеей, поскольку в текущем виде не может быть основой, например, для VLIW процессоров (и VLIW подход в принципе нигде не упоминается). Кроме того, сообщество RISC-V в подавляющем большинстве случаев умалчивает о графических процессорах, обходясь в презентациях туманными отговорками.

4. По сравнению, например, с Power экосистема RISC-V выглядят менее зрело (хоть и более целостно), поскольку реализаций в кремнии в настоящее время у RISC-V всё-таки существенно меньше, а Power уже давно используется в авиации.

5. Заключение

Современные электронные системы стали настолько сложны, что следование открытым стандартам критически важно для жизнеспособности проекта и его экономической целесообразности. Даже при острой необходимости в наличии специфической аппаратной функциональности нет ни одной причины для того, чтобы не рассматривать открытые стандарты как базу для разрабатываемой технологии. RISC-V – хороший и грамотно спроектированный стандарт, позволяющий решать при разработке программно-аппаратных систем такие проблемы как совместимость (в том числе обратная совместимость в долгосрочной перспективе), безопасность, сертификация, энергопотребление, эффективная реализация многопоточности и удешевление разработки. Это важно прежде всего при разработке центральных процессоров и их окружения (интерфейс памяти, кэш и др.), операционных систем, компиляторов, драйверов и высокопроизводительных библиотек. Если вы начинаете новый проект в одной из этих областей – стоит рассмотреть RISC-V.

Среди аналогов достойным конкурентом является система команд Power (и проект OpenPOWER), которая на деле доказала возможность применения единой системы команд в различных приложениях: от встроенных систем до суперкомпьютеров. На него стоит обратить внимание, если для вас важна зрелость технологии, но, возможно, менее важны затраты на стоимость разработки самого чипа или имеется хорошее готовое решение. MIPS недавно также стала открытой благодаря развитию RISC-V, но её недостатки ограничивают расширение. SPARC – очень устаревшая система команд, не оправдавшая себя сразу в нескольких решениях (прежде всего регистровые окна). Кроме того, есть проблемы с преобразованием плавающей точки в целые числа и обратно, поэтому SPARC – в настоящее время это один из самых плохих вариантов.

Остальные рассмотренные нами популярные системы команд – закрытая интеллектуальная собственность коммерческих компаний, только лишь лицензирование которых стоит от 1 миллиона долларов. Доминирующая на сегодняшний день ARM, имея огромный фактический тираж своих чипов (и выигрывая, в основном, дешевизной отдельных экземпляров) до последнего будет сопротивляться переходу мирового сообщества на открытые технологии, но на наш взгляд это лишь вопрос времени. Кроме того, с развитием открытого программного обеспечения необходимость в поддержке x86/x64 в 2020 году исчезающе мала и велика вероятность того, что со временем персональные компьютеры также перейдут на RISC-V, потому что в x86/x64 на сегодняшний день «просто нет смысла» [5].

Список литературы / References

[1]. Tony Albrecht. Pitfalls of Object-Oriented Programming, 2009. Available at: http://harmful.cat-v.org/software/OO_programming/_pdf/Pitfalls_of_Object_Oriented_Programming_GCAP_09.pdf, accessed 01.04.2020.

[2]. Venkataramanan Kumar. [patch][x86_64]: AMD znver2 enablement., 2018. Available at: https://gcc.gnu.org/legacy-ml/gcc-patches/2018-10/msg01982.html?print=anzwix, accessed 01.04.2020.

[3]. Michael Larabel. AMD vs. Intel Contributions To The Linux Kernel Over The Past Decade, 2020. Available at: https://www.phoronix.com/scan.php?page=news_item&px=AMD-Intel-2010s-Kernel-Contrib, accessed 01.04.2020.

[4]. Waterman A.S. Design of the RISC-V instruction set architecture. Electrical Engineering and Computer Sciences, University of California at Berkeley, Technical Report No. UCB/EECS-2016-1, 2016. Available at: https://people.eecs.berkeley.edu/~krste/papers/EECS-2016-1.pdf, accessed 31.03.2020.

[5]. Michael Slater. AMD's K5 Designed to Outrun Pentium. Microprocessor Report, 1994. Available at: http://cgi.di.uoa.gr/~halatsis/Advanced_Comp_Arch/Papers/k5, accessed 31.03.2020.

[6]. Ким А.К., Перекатов В.И., Ермаков С.Г. Микропроцессоры и вычислительные комплексы семейства “Эльбрус”. СПб., Питер, 2013, 272 стр. / Kim A.K., Perekatov V.I., Ermakov S.G. Microprocessors and computing systems of the Elbrus family. St. Petersburg, Peter, 2013, 272 p.
[7]. Hugh Blemings. Final Draft of the Power ISA EULA Released, 2020. Available at: https://openpowerfoundation.org/final-draft-of-the-power-isa-eula-released/, accessed 31.03.2020.
[8]. Raptor Computing Systems. Talos II – the world's first computing system to support the new PCIe 4.0 standard – also boasts substantial DDR4 memory, dual POWER9 CPUs., 2019. Available at: https://www.raptorcs.com/TALOSII/, accessed 31.03.2020.
[9]. Anton Blanchard, Paul Mackerras. Microwatt project on GitHub. A tiny Open POWER ISA softcore written in VHDL 2008, 2020. Available at: https://github.com/antonblanchard/microwatt, accessed 31.03.2020.
[10]. Luke Kenneth Casson Leighton, Yehowshua Immanuel, Jacob Lifshay and others. Libre-RISCV GPU project., 2019. Available at: https://libre-riscv.org/3d_gpu/, accessed 31.03.2020.
[11]. Luke Kenneth Casson Leighton, [libre-riscv-dev] power pc, 2019. Available at: http://lists.libre-riscv.org/pipermail/libre-riscv-dev/2019-October/003035.html, accessed 31.03.2020.
[12]. Open Hardware GNU/Linux PowerPC notebooks, 2020, Available at: https://www.powerpc-notebook.org/en/, accessed 31.03.2020.
[13]. GCC Manual. , 6.49. Using Vector Instructions through Built-in Functions, 2019, Available at: https://gcc.gnu.org/onlinedocs/gcc-4.7.2/gcc/Vector-Extensions.html, accessed 01.04.2020.
[14]. Крсте Асанович, Александр Редькин и др. Технический симпозиум RISC-V Moscow, 2019 / Krste Asanovic, Alexander Redkin and others. Technical Symposium RISC-V Moscow, 2019. Available at: https://riscv.expert/, accessed 01.04.2020.

Информация об авторах / Information about authors

Владимир Александрович ФРОЛОВ – кандидат физико-математических наук, старший научный сотрудник Института прикладной математики им. М.В. Келдыша РАН, научный сотрудник факультета Вычислительной математики и кибернетики Московского университета. Сфера научных интересов: реалистичная компьютерная графика, моделирование освещённости, разработка программных систем оптического моделирования, параллельные и распределённые вычисления.

Vladimir FROLOV – PhD in computer graphics, senior researcher in the Keldysh Institute of Applied Mathematics of Russian Academy of Sciences and researcher in computer graphics of Moscow State University. Research interests: realistic computer graphics, light transport simulation, elaboration of optical simulation software systems, GPU computing.

Владимир Александрович ГАЛАКТИОНОВ – доктор физико-математических наук, профессор, заведующий отделом компьютерной графики и вычислительной оптики Института прикладной математики им. М. В. Келдыша РАН с 2003 г. Сфера научных интересов: компьютерная графика, оптическое моделирование, создание программных систем оптического моделирования.

Vladimir GALAKTIONOV – Doctor of Science in physics and mathematics, Professor, Head of Computer graphics department in the Keldysh Institute of Applied Math of RAS since 2003. Research interests: computer graphics, optical simulation, elaboration of optical simulation software.

Вадим Владимирович САНЖАРОВ – младший научный сотрудник факультета Вычислительной математики и кибернетики Московского университета. Сфера научных интересов: компьютерная графика, системы фотореалистичного синтеза изображений, параллельное и распределённое программирование.

Vadim SANGAROV – junior researcher in computer graphics at Moscow State University. Research interests: realistic computer graphics, elaboration of optical simulation software systems, concurrent and distributed computing.

DOI: 10.15514/ISPRAS-2020-32(2)-8

Hard drives monitoring automation approach for Kubernetes container orchestration system

A.S. Shemyakinskaya, ORCID: 0000-0003-1739-7748 <shem98a@mail.ru>
I.V. Nikiforov, ORCID: 0000-0003-0198-1886 <igor.nikiforovv@gmail.com >
Peter the Great St.Petersburg Polytechnic University,
195251, Russia, St.Petersburg, Polytechnicheskaya, 29

Abstract. Today, a laborious and non-trivial task is to automate monitoring of hard drives in a cluster infrastructure using the Kubernetes container management system. The paper discusses existing approaches to monitoring hard drives in the Kubernetes container orchestration system and provides a comparative analysis of them. Based on the presented analysis, a conclusion is drawn on the need to improve and automate approaches. The paper proposes an approach to automating the collection of metrics from hard drives by implementing the Kubernetes "operator" for a tool with which you can effectively obtain information about the state of hard drives in the system. As results, the temporal characteristics of collecting information about disks using existing approaches and the proposed approach are given. Numerical results and graphs showing the gain of the proposed approach are presented

Keywords: Monitoring of hard drives; Kubernetes Operator; Container Orchestration System

For citation: Shemyakinskaya A.S., Nikiforov I.V. Hard drives monitoring automation approach for Kubernetes container orchestration system. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 99-106. DOI: 10.15514/ISPRAS–2020–32(2)–8

Подход автоматизации мониторинга дисковых носителей для системы оркестрации контейнеров Kubernetes

А.С. Шемякинская, ORCID: 0000-0003-1739-7748<shem98a@mail.ru>
И.В. Никифоров, ORCID: 0000-0003-0198-1886 <igor.nikiforovv@gmail.com>
Санкт-Петербургский Политехнический Университет Петра Великого,
195251, Россия, Санкт-Петербург, ул. Политехническая, д. 29.

Аннотация. На сегодняшний день трудоемкой и нетривиальной задачей является автоматизация мониторинга дисковых носителей в кластерной инфраструктуре при использовании системы управления контейнерами Kubernetes. В работе рассматриваются существующие подходы к мониторингу дисковых носителей в системе оркестрации контейнеров Kubernetes и приведен их сравнительный анализ. На основе представленного анализа делается вывод о необходимости совершенствования и автоматизации подходов. Для этого в работе предлагается подход автоматизации сбора метрик с дисковых носителей за счет реализации Kubernetes «оператора» для инструмента, с помощью которого можно эффективно получать информацию о состоянии дисковых носителей в системе. В качестве результатов приведены временные характеристики сбора информации о дисках с использованием существующих подходов и предлагаемого подхода. Приведены численных результаты и графики, демонстрирующие выигрыш предлагаемого подхода.

Ключевые слова: мониторинг дисковых носителей; Kubernetes Operator; система оркестрации контейнеров

Для цитирования: Шемякинская А.С., Никифоров И.В. Подход автоматизации мониторинга дисковых носителей для системы оркестрации контейнеров Kubernetes. Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 99-106 (на английском языке). DOI: 10.15514/ISPRAS–2020–32(2)–8

1. Introduction

Currently container orchestration systems for the automatic deployment of applications in a cluster infrastructure are gaining popularity. These systems include Kubernetes [1], Docker Swarm [2], Mesos [3].

At the same time a lot of applications need to store their data on hard drives, which sets in turn increasing requirements for the reliability of used hard drives. That's why special attention is paid to monitoring of hard drive state in a cluster infrastructure to prevent their failure and data loss. Many companies such as Dell EMC encounter the problem of disk monitoring.

System administrators and engineers who monitor hard drives in a cluster system use different methods and approaches for obtaining disk information, discussed in Section 2.

Section 3 provides a theoretical basis and defines container orchestration systems using Kubernetes as an example. Section 4 describes the operator pattern for Kubernetes.

Section 5 describes advantages of the proposed approach for automating disk monitoring in a Kubernetes cluster.

Section 6 covers the implementation of the proposed approach using the "operator" for disk monitoring tool.

Section 7presents the results and temporal characteristics of collecting information about disks using existing approaches and the proposed one. Charts show the time gain while using the proposed approach.

2. Approaches for evaluation of the hard drives statuses

One way to get the state of hard drive is to use utilities built into the Linux operating system such as lsblk, fdisk, etc. lsblk utility [4] displays information about all available or specified device blocks. It reads information from the sysfs and udev db file systems. The problem with this approach is that the lsblk utility does not provide information about the current state and availability of the disk.

Another disk health evaluation utility is smartctl. This Linux utility allows to retrieve S.M.A.R.T. disk information. S.M.A.R.T. is a technology which allows to analyze and predict the state of hard drives [5]. S.M.A.R.T. monitors the main characteristics of the drive. Each receives estimates and then recounts them into numbers. Depending on the reference value, the state of the disk can be estimated from the result. There are over 150 S.M.A.R.T. indicators with their own reference values, so manual analysis of such information is quite time-consuming for a person. To reduce the complexity of the analysis S.M.A.R.T. indicators, neural networks can be used, which is described in [6, 7].

There is also an intelligent platform management interface (IPMI) designed for remote monitoring and control of a computer system [8]. It is possibly to connect remotely to the server and manage its operation using IPMI. The IPMI specification standardizes an interface. Various companies have the implementations of this interface: IDRAC (DELL), Cisco IMC (Cisco), ILO (HP). This approach is more efficient than using lsblk and smartctl utilities, because specific IPMI implementations (IDRAC, IMC, ILO) provide a graphical interface and a wide range of server functionality in addition to monitoring hard drives. The disadvantage of this method is the lack of free licenses for products that implement the IPMI interface.

The last of the considered approaches is the Linux ipmitool command [9], which implements the interface, but its functionality is limited by the ability to provide information about FRU, LAN configuration, sensor readings and remote power management. The hardware also must have a special BMC port in order to use IPMI, so this approach is not universal.

Considering the approaches, the automating analysis of the state of hard drive through the implementation of Kubernetes «operator» of the disk monitoring tool is proposed. A more detailed definition of the «operator» of Kubernetes is given in Section 4.

The proposed approach is better than others, because of efficient and automatic provision of information on the current state of disks without creating additional load on the Kubernetes cluster.

3. Container orchestration systems

Containerization is an approach in software development which allows an application or service, its dependencies and configuration (abstract deployment manifest files) to be packaged together into a container image [10]. In fact, this is virtualization at the operating system level. Containers greatly simplify and automate application deployment regardless of environment. In turn, the Docker tool [11] helps simplify the creation and launch of containers. As modern applications include more and more containers and distributed servers require complex management and deployment of applications, there is a need for container orchestration systems.

Container orchestration allows to determine how to select, deploy, track and dynamically manage the configuration of multi-container packaged applications [12].

Container orchestration concerns not only the initial deployment of multi-container applications, but also includes management, for example, scaling a multi-container application as a single object.

The most popular is the Kubernetes container orchestration system [1]. Kubernetes is the open source software needed to manage and deploy a Docker container cluster.

A Kubernetes deployment is known as a cluster. Kubernetes cluster can be imagined as two parts: a management layer, which consists of the main node(s) and worker nodes. Pods consisting of containers are made on working nodes. Each node represents its own Linux environment and can be either a physical or virtual machine.

4. Pattern «operator»

«Operator» is a Kubernetes plug-in that uses user resources to manage applications and their components [13].

A resource in Kubernetes stores a collection of specific types of API objects. Kubernetes objects are persistent resources in the Kubernetes system. Each resource has HTTP request at a unique address, which is processed by the server with the Kubernetes API and returns information about this resource. Kubernetes uses these entities to represent the state of a cluster. For example, Kubernetes Deployment is an object that can represent an application running in a cluster. While deploying applications, there should be a specification for the final configuration. For example, setting up three replicas of an application makes Kubernetes system reads the specification and starts three instances of the desired application, updating the state according to the configuration.

Pods can be example of resources. It is embedded resource and it contains a collection of pod objects. Custom Resource is a Kubernetes API extension that is not necessarily available when Kubernetes is installed by default.

«Operator» combines user resources and custom controllers. They allow to monitor the specified resources and maintain their state in accordance with the value the user set. When a corresponding event occurs with a resource, the “operator” reacts and performs a specific action.

5. Disk information gathering automating approach using «operator»

Fig. 1 shows the scheme of the disk information gathering automating approach in the Kubernetes cluster and user interaction.

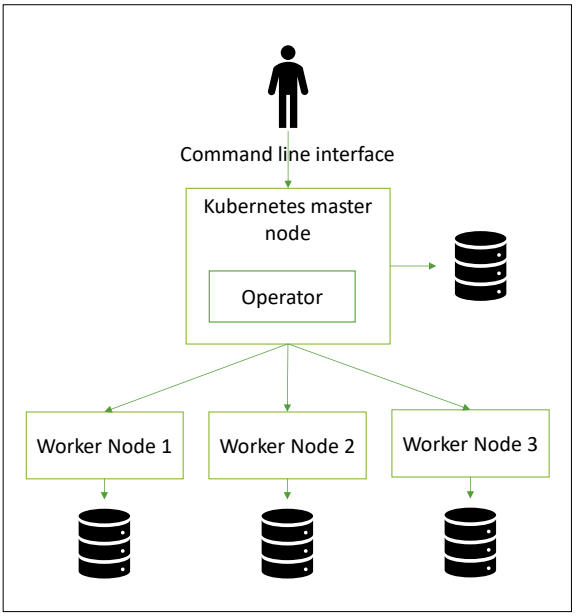

Fig. 1. Scheme of the "the disk information gathering automating approach " in the Kubernetes cluster

The approach proposes to introduce an «operator» into the Kubernetes container orchestration system to automate monitoring of hard drives on nodes. «Operator» is an application deployed on a Kubernetes cluster. It works with its Kubernetes API and monitors events within the cluster. Its task is to collect information from cluster nodes about hard drives, create disk objects on the cluster and monitor them.

```
$ operator-sdk new monitoring-operator --repo
github.com/example-inc/monitoring-operator
$ cd monitoring-operator

$ operator-sdk add api --api-
version=app.example.com/v1alpha1 --kind=Disk

$ operator-sdk add controller --api-
version=app.example.com/v1alpha1 --kind=Disk
```

Fig. 2. Command for generating code template

Once objects are created, Kubernetes can provide information about them through a client application. Using the command line utility, the user can access the Kubernetes cluster to obtain information about the specified resources. The use of the «operator» has several advantages.

- «Operator» allows to take advantage of Kubernetes, e.g. work with custom resources.
- One of the advantages of the “operator” is the ability to process the network events of the Kubernetes cluster (GET, CREATE, DELETE, PATCH, etc.), that is, the programmer decides what should happen to the custom resource during an event.
- Since a resource is created in the system, this opens the possibility of using the Kubernetes command-line interface. The user can get information about all resources with a single command (Fig. 2).

- The information that Kubernetes can show about resources also might be regulated. For example, if you need to know the status of disks, then the command line can show only the requested information. This filtering approach reduces human analysis time.
- Kubernetes provides data in the form of a table, which simplifies the analysis of information by a person.
- Kubernetes administrators can create volumes, knowing which drives can be used for this.

Thus, the use of the «operator» allows not only to automate monitoring of hard drives, but also to reduce the time of analyzing disk information. Working with the Kubernetes API allows to take advantage of Kubernetes described above.

6. «Operator» implementation

6.1 Structure of «operator» units

Fig. 3 shows the architecture of the «operator» for disk monitoring.

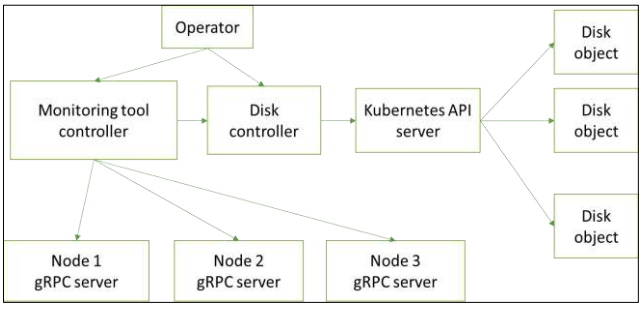

Fig. 3. The architecture of the «operator»

«Operator» is a software tool for Kubernetes, which implements two controllers: monitoring tool controller and disk resource controller.

The controller of the monitoring tool deploys a server on each node of the cluster, which collects information about the hard drives of this node on request.

The disk resource controller requests this information and sends a request to create a resource in Kubernetes. Kubernetes API server based on the received information creates custom resources (CR) for disks. Disks CR store information about a specific drive from a node, for example, serial number, size, etc.

6.2 Technical implementation features

Since all Kubernetes «operators» have the same structure, it is possible to generate common code, filling it with own logic. Usually such tools are used for generation as Operator-SDK [14], Kubebuilder [15], Code-Generator [16] and written. Usually Golang programming language is used for «operator» development

The Operator-SDK tool was used in this work. It allows to generate a resource structure template for Kubernetes, in the case of an “operator” for disk monitoring - Disk Custom Resource Definition (CRD) and a resource represented disk monitoring tool on each machine in the Kubernetes cluster. It runs as a gRPC server. To generate code template Operator-sdk has a command (fig. 3). Controllers have also been created to manage these resources. Each controller has a synchronization function (Reconcile loop). It is called by the Kubernetes system every time during any events happening to a custom resource.

- First command generates Golang project for «operator».
- Second command generate Golang structure represented Kubernetes objects (fig. 4).

- Third command generate code template for object controller with Reconcile loop.

```
type DiskSpec struct {
        DiskID          string
`json:"disk_id"`
        NodeID          string
`json:"node_id"`
        Path            string
`json:"path"`
        Capacity        int
`json:"capacity"`
        SerialNumber    string
`json:"serialNumber"`
        DriverHealth    string
`json:"driveHealth"`
}
```

Fig. 4. Disk structure in Golang

In the operator implementation, the controller of the disk monitoring tool in the Reconcile loop ensures that the gRPC server is running on each node of the cluster. In turn, the Disk CRD controller in the synchronization function creates a connection with the monitoring servers and makes a request for information about the disks. Then using a REST client, the request for creating resources is sent. Based on the information received, Kubernetes creates disk object. If such object already exists, then Kubernetes updates it. The synchronization cycle is called every 2 seconds to keep disk information up to date.

To build docker image the command on fig. 5 can be used.

```
operator-sdk build operator:v0.0.1
```

Fig. 5. Command for building docker image

To deploy «operator» in Kubernetes cluster Helm tool was used [17]. This tool allows to deploy application using one command (fig. 6). We can develop special manifests (charts) for Helm to represent «operator» application in Kubernetes. These manifests contain all necessary information about the application, so Kubernetes deploy it correctly. For example, we can specify how many replicas of application should be deployed on Kubernetes cluster.

```
helm install operator charts/operator
```

Fig. 6. Command for deploying "operator" in Kubernetes cluster

Thus, Kubernetes has a resource that can be accessed using the command line or an HTTP request. Output is shown in fig. 7.

```
~# kubectl get disks
NAME              NODE     HEALTH
disk-node1-dev-sda  node1  HEALTH_GOOD
disk-node2 -dev-sdb  node2  HEALTH_GOOD
disk- node3-dev-sdc  node3  HEALTH_UNKNOWN
disk- node4 -dev-sdb  node4  HEALTH_FAILED
```

Fig. 7. Command line output

7. Results

Table 1 shows the results of assessing the status of the Kubernetes cluster hard drives in three ways:

- using the smartclt utility;
- using the IDRAC program (implementation of the ipmi interface);

- the proposed approach with the “operator” Kubernetes.

The characteristics of the tested cluster are:

- the cluster consists of 4 nodes;
- each node has 4 hard drives (system drive: 112 GB SSD, 3 drives: 8 TB HDD)
- each node has a Linux operating system deployed.

The necessary command was entered on each node in order to obtain information.
Measurements were performed three times on the cluster. It was made of the time to enter the necessary tool commands to obtain information about the disks. The time of manual analysis of the output of commands is also measured. The average 3
value of each measurement was taken. In the future, it is planned to automate this process and collect more metrics.

Table 1. Comparative analysis of disk monitoring approaches

Time\Approach	Smartctl utility	ipmi (IDRAC)	Using operator
T_c (seconds)	5	7	5
T_d (seconds per disk)	10	2	2
T_a (second) in tested system	60	36	28

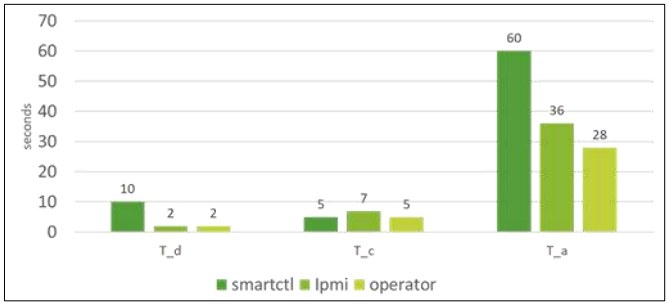

Fig. 8. Chart of spending time on analysis for various instruments

T_d expresses the time for manual analysis of the received logs from the specified methods.

T_c is the time to enter commands into the terminal.

T_a – the total time spent in the tested system. This is the time to enter the command and the time to analyze the logs.

We can conclude from table 1 and the graph in fig. 8, the gain of the “operator” approach compared to IDRAC was 8 seconds, and for smartctl utility the total analysis time decreased by 2 times.

7. Conclusion

Thus an “operator” approach was developed for Kubernetes, which provides information about hard drives in a cluster infrastructure.

This approach requires 2 times less time than using built-in utilities in the Linux operating system. The proposed approach is suitable for different platforms, as it runs in a Kubernetes cluster, which can be run on different platforms.

However, the approach has several disadvantages compared to the existing ones. For example, the smartctl utility and the ipmi IDRAC implementation provide more information about hard drives, while the “operator” provides only status.

In the future, it is planned to expand the list of indicators that can be analyzed through the kubectl utility using the "operator" and to create a graphical interface and scripts to automate command input to reduce the amount of time analyzing disk information.

References / Список литературы

[1] Kubernetes, Available at: https://kubernetes.io/, accessed 20.05.2020.

[2] Docker Swarm, Available at: https://docs.docker.com/engine/swarm/, accessed 20.05.2020..

[3] Mesos, Available at: http://mesos.apache.org/, accessed 20.05.2020.

[4] Adam K.D. Linux Administration Cookbook. Birmingham, UK, Packt Publishing, 2016, 783 p.

[5] A. Chatzidimitriou, G. Papadimitriou, and D. Gizopoulos. HealthLog Monitor: Errors, Symptoms and Reactions Consolidated. IEEE Transactions on Device and Materials Reliability, vol. 19, no. 1, 2019, pp. 46-54.

[6] S. Ganguly et al. A Practical Approach to Hard Disk Failure Prediction in Cloud Platforms: Big Data Model for Failure Management in Datacenters. In Proc of the 2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService), 2016, pp. 105-116.

[7] X. Sun et al. System-level hardware failure prediction using deep learning. In Proc of the 2019 56th ACM/IEEE Design Automation Conference (DAC), 2019, pp. 1-6.

[8] IPMI Specification. Available at: https://www.intel.ru/content/www/ru/ru/products/docs/servers/ipmi/ipmi-second-gen-interface-spec-v2-rev1-1.html, accessed 20.05.2020.

[9] ipmitool man page. Available at: https://linux.die.net/man/1/ipmitool, accessed 20.05.2020.

[10] M. Abdelbaky et al. Docker Containers across Multiple Clouds and Data Centers. 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing, 2015, pp. 368-371.

[11] Docker. Available at: https://www.docker.com/, accessed 20.05.2020.

[12] I.M.A. Jawarneh et al. Container Orchestration Engines: A Thorough Functional and Performance Comparison. In Proc of the 2019 IEEE International Conference on Communications (ICC), 2019, pp. 1-6.

[13] Operator Pattern. Available at: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/, accessed 20.05.2020.

[14] Operator SDK. Available at: https://github.com/operator-framework/operator-sdk, accessed 20.05.2020.

[15] Kubebuilder. Available at: https://github.com/kubernetes-sigs/kubebuilder, accessed 20.05.2020.

[16] Code-generator. Available at: https://github.com/kubernetes/code-generator, accessed 20.05.2020.

[17] Helm. Available at: https://helm.sh/, accessed 20.05.2020.

Information about authors / Информация об авторах

Anastasia Sergeevna SHEMYAKINSKAYA is a 4-th year student of Institute of computer science and technologies of Peter the Great St.Petersburg Polytechnic University. Research interests are container orchestration systems, containerization, cloud computing.

Анастасия Сергеевна ШЕМЯКИНСКАЯ – студентка 4 курса Института компьютерных наук и технологий СПбПУ. Сферы научных интересов: системы оркестровки контейнеров, контейнеризация, облачные вычисления.

Igor Valerevich NIKIFOROV is a candidate of engineering sciences, associate professor of High school of software engineering, Institute of computer science and technologies of Peter the Great St.Petersburg Polytechnic University. His research interests include big data.

Игорь Валерьевич НИКИФОРОВ – кандидат технических наук, доцент Высшей школы программной инженерии Института компьютерных наук и технологий СПбГУ. Сфера научных интересов: большие данные.

DOI: 10.15514/ISPRAS-2020-32(2)-9

Верифицированная тактика Isabelle/HOL для теории ограниченных целых на основе инстанцирования и SMT

[1]*Р.Ф. Садыков, ORCID: 0000-0002-2792-2465 <sadykov@ispras.ru>*
[2] *М.У. Мандрыкин, ORCID: 0000-0002-9306-7719 <mandrykin@ispras.ru>*
[1] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*
[2] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*

Аннотация. При дедуктивной верификации Си-программ как с помощью различных платформ верификации (Why3, Frama-C/WP, F*), так и с помощью систем интерактивного доказательства теорем (Isabelle, HOL4, Coq) достаточно широко используются SMT-решатели. Их разрешающие алгоритмы полны для некоторых комбинаций логических теорий (логик), в частности для логики QF_UFLIA. В то же время при верификации Си-программ часто необходимо оперировать формулами в других разрешимых логиках, поддерживаемых не всеми SMT-решателями. Типичными примерами таких логик могут служить комбинации QF_UFLIA с теориями ограниченных целых как с переполнением (для беззнаковых целых в Си), так и без переполнения (для знаковых целых), а также теорией конечных интерпретируемых множеств (для поддержки рамочных условий) и др. Одним из возможных способов поддержки таких логик является их непосредственная реализация в SMT-решателях, однако этот способ часто является трудоемким, а также недостаточно гибким и универсальным. Другим способом является реализация пользовательских стратегий инстанцирования кванторов для сведения формул в неподдерживаемых логиках к формулам в широко распространенных разрешимых логиках, таких как QF_UFLIA. В данной статье представлена процедура инстанцирования лемм для трансляции формул в теории ограниченных целых без переполнения в логику QF_UFLIA. Для процедуры трансляции даны доказательства корректности и полноты, а также описана формализация этих доказательств в системе Isabelle/HOL. Аналогичный подход можно использовать для формулирования и доказательства полноты процедур трансляции формул в других теориях, таких как теория ограниченных целых с переполнением и теория ограниченной адресной арифметики.

Ключевые слова: статическая верификация; задача выполнимости формул в теориях; автоматизированные процедуры принятия решений

Для цитирования: Садыков Р.Ф., Мандрыкин М.У. Верифицированная тактика Isabelle/HOL для теории ограниченных целых на основе инстанцирования и SMT. Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 107-124. DOI: 10.15514/ISPRAS–2020–32(2)–9

Благодарности. Исследование поддержано Министерством образования и науки РФ (проект №RFMEFI60719X0295).

Verified Isabelle/HOL tactic for the theory of bounded integers based on quantifier instantiation and SMT

[1] *R. Sadykov ORCID: 0000-0002-2792-2465 <sadykov@ispras.ru>*
[2] *M. Mandrykin ORCID: 0000-0002-9306-7719 <mandrykin@ispras.ru>*
[1] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia
[2] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. SMT solvers are widely applied for deductive verification of C programs using various verification platforms (Why3, Frama-C/WP, F*) and interactive theorem proving systems (Isabelle, HOL4, Coq) as the decision procedures implemented in SMT solvers are complete for some combinations of logical theories (logics), in particular for the QF_UFLIA logic. At the same time, when verifying C programs, it is often necessary to discharge formulas in other logical theories and their combinations, that are also decidable but not supported by all SMT solvers. Theories of bounded integers both with overflow (for unsigned integers in C) and without overflow (for signed integers), and also theory of finite interpreted sets (needed to support frame conditions) are good examples of such theories. One of the possible ways to support such theories is to directly implement them in SMT-solvers, however, this method is often time-consuming, as well as insufficiently flexible and universal. Another way is to implement custom quantifier instantiation strategies to reduce formulas in unsupported theories to formulas in widespread decidable logics such as QF_UFLIA. In this paper, we present an instantiation procedure for translating formulas in the theory of bounded integers without overflow into the QF_UFLIA logic. We formally proved soundness and completeness of our instantiation procedure in Isabelle. The paper presents an informal description of this proof as well as some considerations on the efficiency of the proposed procedure. Our approach is sufficient to obtain efficient decision procedures implemented as Isabelle/HOL proof methods for several decidable logical theories used in C program verification by relying on the existing capabilities of well-known SMT solvers, such as Z3 and proof reconstruction capabilities of the Isabelle/HOL proof assistant.

Keywords: static verification; quantifier instantiation; SMT formulas; SMT solvers; automated decision procedures; software verification

For citation: Sadykov R., Mandrykin M. Verified Isabelle/HOL tactic for the theory of bounded integers based on quantifier instantiation and SMT. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 107-124 (in Russian). DOI: 10.15514/ISPRAS–2020–32(2)–9

Acknowledgement. The research was carried out with funding from the Ministry of Science and Higher Education of the Russian Federation (the project unique identifier is RFMEFI60719X0295).

1. Введение

В процессе разработки программного обеспечения на языке Си часто возникают ошибки, связанные с неправильным использованием арифметических операций, которые могут привести к переполнению, и, как следствие, непредвиденному поведению программ, несмотря на долгое и тщательное тестирование. Но благодаря формальным методам верификации программного кода возможно найти неочевидные, но в то же время многочисленные ошибки, связанные с операциями над целочисленными типами и их значениями. Многие методы и инструменты дедуктивной верификации используют решатели задачи выполнимости формул в теориях, называемые также SMT-решателями. Алгоритмы таких решателей полны для некоторых комбинаций логических теорий, в частности, для логики QF_UFLIA. Несмотря на применимость SMT-решателей к широкому классу задач, часто необходимо применять решающие процедуры к формулам в логических теориях, которые являются разрешимыми, но не всегда полностью поддерживаются SMT-решателями. К таким логическим теориям можно отнести ограниченные целые как с переполнением (для беззнаковых целых в Си), так и без переполнения (для знаковых целых), теории конечных интерпретируемых множеств (для поддержки рамочных условий) и др. В

данной работе с целью автоматизированного получения доказательств в системе интерактивного доказательства теорем Isabelle/HOL [1] мы сформулировали метод проверки выполнимости формул с ограниченными целочисленными значениями переменных, который основан на применении SMT-решателя в логике QF_UFLIA. Наш метод является решающей процедурой над формулами в теории ограниченных целых (BLIA), которая реализована в виде метода Isabelle/HOL и состоит из инстанцирования исходной формулы аксиомами теории ограниченных целых (являющихся леммами теории HOL-Word[2]), интерпретации полученной формулы в теории QF_UFLIA, применения решающей процедуры SMT-решателя и последующего восстановления доказательства средствами Isabelle. В свою очередь, доказательство полноты предложенного метода основывается на преобразовании полученной от SMT-решателя модели в логике QF_UFLIA в модель в теории ограниченных целых.

Несмотря на инстанцирование аксиом исходной теории, которое увеличивает длину анализируемой формулы, нам не требуется вносить существенные изменения в инструменты воспроизведения доказательств системы Isabelle и SMT-решатель, а благодаря оптимизации аксиом теории ограниченных целых мы получаем линейное по количеству вхождений функциональных символов увеличение длины исходной формулы, поэтому сложность метода не существенно увеличивается по сравнению с использованием решающих процедур, реализованных непосредственно внутри SMT-решателей. Основной целью нашей работы являлось доказательство полноты и корректности предложенной процедуры преобразования формул из теории ограниченных целых в логику QF_UFLIA.

Также мы рассмотрели некоторые статьи о модульной арифметике [3], [4], [5], где описаны ошибки с переполнениями переменных типа int и верификация программ, использующих ограниченные целые. В работе [3] сформулированы методы преобразования для формул в нелинейной модульной арифметике с ограниченными целыми, которые предполагают изменение внутренних алгоритмов SMT-решателя. Потенциально получаемые при этом доказательства (соответствующая дедуктивная система подробно не описана в статье), иногда сложно воспроизвести с использованием существующих возможностей систем автоматизированных доказательств. В статье [4] получена оценка сложности решающей процедуры для IDL. IDL – это логика разности над целыми числами. По определению, это булевы комбинации неравенств вида $x - y < b$, где x и y – целочисленные переменные, а b – целочисленная константа. Решающая процедура данной логики является NP-трудной; аналогичная логика QF_UFLIA, которая является расширением логики IDL операцией умножения на константу, – тоже NP-полная задача.

2. Предварительная информация

В стандарте SMT-LIB [8] QF_UFLIA – это логика бесквантворных формул с неинтерпретируемыми символами и равенством в комбинации теорий линейной целочисленной арифметики и неинтерпретируемых функций. Логику QF_UFLIA можно рассматривать как комбинацию логик QF_LIA и QF_UF. QF_LIA обозначают замкнутые бесквантворные формулы с равенством в теории линейной целочисленной арифметики (LIA). В сигнатуру этой теории входят следующие функциональные символы: $\{+, c \times, \cdot, \leq\}$, где c – целочисленная константа. QF_UF обозначает замкнутые бесквантворные формулы с равенством в теории неинтерпретируемых функций.

Общеизвестным методом решения задачи доказательства условий корректности с ограниченными целыми числами является представление целых чисел в виде битовых векторов. Но как написано в статье [11], логика QF_BV очень сложная в общем случае, на практике различие во времени разрешения формул в QF_BV и QF_LIA часто выглядит как экспоненциальное. Кроме того, сложность решения в QF_BV зависит от размера ограниченного целого, поэтому для больших ограниченных целых, например, 256 бит,

различие эффективности между QF_BV и QF_LIA может быть очень большим. Для QF_LIA размер никак не влияет на время решения. Для QF_BV нет поддержки воспроизведения доказательств в Isabelle. И доказательства обычно больше по сравнению с LIA, так как основаны на переборе большого числа вариантов и не переиспользуют встроенную процедуру для линейной арифметики в Isabelle. Но есть фрагменты логики QF_BV, которые можно решать существенно более эффективно, но в решателях реализованы не эти алгоритмы, а общий случай. Эффективную решающую процедуру для линейного фрагмента со сдвигами и побитовыми операциями можно тоже сделать на основе процедуры конечного инстанцирования, аналогично как в этой работе.

Теорию линейной арифметики с ограниченными целыми мы будем задавать аксиоматически как расширение теории линейной целочисленной арифметики. В ее сигнатуру помимо символов теории LIA включим следующие функциональные символы: $\{\cdot +_b \cdot, c \times_b \cdot, v(\cdot), (\cdot)_b, \leq_b \cdot\}$, где c – неограниченная целочисленная константа. Термы в этой теории могут быть двух различных типов – ограниченные и неограниченные целые. Ограниченные целые принимают целочисленные значения из некоторого диапазона $[L..U]$. Эти значения являются неограниченными целыми и могут быть получены с помощью применения функции $v(\cdot)$. Будем использовать a, b и d в качестве переменных, относящихся к сорту ограниченных целых. Для сорта неограниченных целых будем использовать переменные x, y и c, при этом c всегда будет обозначать интерпретируемую целочисленную константу. Таким образом, для любого ограниченного целого a будет выполнено соотношение $L \leq v(a) \leq U$. Семантика операций $\cdot +_b \cdot$ и $\cdot \times_b \cdot$ такова, что их результаты совпадают с результатами соответствующих целочисленных операций, примененных к значениям ограниченных аргументов, если эти результаты могут быть представлены в виде ограниченных целых. В противном случае эти результаты не определены, то есть их модель не фиксирована и может быть выбрана произвольно для каждой конкретной формулы. Операция $\cdot_b$ возвращает ограниченное целое с заданным значением, если это значение лежит в диапазоне $[L..U]$ и имеет неопределенный результат в противном случае. Операция $\cdot \leq_b \cdot$ сравнивает значения ограниченных целых и по сути является сокращенным обозначением для выражения вида $v(\cdot) \leq v(\cdot)$. В качестве примеров формул в теории ограниченных целых рассмотрим следующие:

$$v(a) \leq 5 \wedge 5 \leq v(a) \wedge (5)_b \times_b a +_b (1)_b \neq (26)_b \text{ и} \quad (1)$$

$$a \leq_b (5)_b \wedge (5)_b \leq_b a \wedge (5 \times v(a) + 1 - 26)_b \neq (0)_b. \quad (2)$$

Будем считать $L = 0$ и $U = 25$. В этих предположениях формула (1) будет являться выполнимой, например, если взять в качестве значения $v(a)$ ограниченной переменной a число 5 ($v(a) = 5$, $(5)_b \times_b a = (25)_b$, но $(5)_b \times_b a +_b (1)_b$ и $(26)_b$ не определено, так как $25 + 1 = 26 > 25$ и может быть выбрано равным, например, $(0)_b$). Формула (2) же является не выполнимой, так как из $a \leq_b (5)_b \wedge (5)_b \leq_b a$ следует, что $v(a) = 5$, а значит $5 \times v(a) + 1 - 26 = 0$ и $(0)_b = (0)_b$.

Для решения задачи о выполнимости формул в теории ограниченных целых будем использовать процедуру трансляции в логику QF_UFLIA. Процедура преобразования формулы F состоит из последовательного инстанцирования аксиом теории ограниченных целых. Инстанцированием аксиомы для некоторой формулы F будем называть взятие конъюнкции от аксиомы с некоторой подстановкой и формулы F. Операция инстанцирования аксиомы довольна проста, мы применяем инстанцирование, когда можем найти подходящий терм в формуле F согласно правилам процедуры преобразования. Такое преобразование создает некоторую формулу F^*, которую мы интерпретируем в теории QF_UFLIA, и запускаем решающую процедуру SMT-решателя, получая доказательство, которое затем воспроизводится (сертифицируется) средствами Isabelle [6], [7]. Для доказательства полноты процедуры преобразования рассматриваем случай, когда мы получили некоторую модель R преобразованной формулы F^* в логике QF_UFLIA. Далее будем называть модель R реализацией формулы F. В данной статье мы показываем, как из

реализации R может быть восстановлена модель M исходной формулы F в теории ограниченных целых. В следующих разделах рассмотрены основные обозначения и определения, описывающие данную процедуру, пример работы алгоритма, формулировка и доказательство соответствующей теоремы о полноте.

3. Постановка задачи

Пусть $\mathbb{Z}$ – множество целых чисел, $\mathbb{Z}_b$–множество ограниченных целых и $\sum = \{+_b, \times_b, \leq_b\}$ – сигнатура теории T линейной арифметики с ограниченными целыми (BLIA). F – формула в BLIA, в которой переменные принимают значения из множества $\mathbb{Z}_b$.

Множество аксиом теории ограниченных целых состоит из аксиом линейной целочисленной арифметики и 5 формул (см. рис. 1). Обозначим это множество за T_0, где L и U являются ограничениями на целые, символ $+_b$ обозначает операцию суммирования в $\mathbb{Z}_b$, $\times_b$ – умножение в $\mathbb{Z}_b$, функция v отображает из множества $\mathbb{Z}_b$ в $\mathbb{Z}$. В практической реализации нашего инструмента в Isabelle/HOL мы рассматриваем эти 5 формул как леммы теории «HOL-Word» [2]. Так как аксиома $A6$ в теории T_0 зависит от двух переменных a и b, мы вводим расширение теории T_0 – теорию T_1, чтобы избежать квадратичного увеличения размера формулы с инстанцированными аксиомами. Поэтому наш алгоритм реализует процедуру инстанцирования аксиомами из теории T_1.

Теория T_1 отличается от T_0 аксиомой $A6$ (см. аксиому $A6'$ на рис. 1).

Покажем эквивалентность теорий T_0 и T_1.

Теорема 1. *Каждая формула F без свободных переменных в сигнатуре* $\sum$ *выполнима в теории* T_0 *тогда и только тогда, когда она выполнима в теории* T_1.

Доказательство. Покажем, что для любой формулы F, имеющей модель M в теории T_0, существует та же модель M в теории T_1 и наоборот.

Пусть формула F имеет модель M в теории T_0. Тогда все аксиомы, кроме ($A6'$), содержатся в M. Возьмем произвольное ограниченное целое a. Поскольку ($A4$) сохраняется, мы имеем

$$L \leq v^M(a) \leq U.$$

Обозначим $v^M(a)$ как c. Тогда $L \leq c = v^M(a) \leq U$, и поскольку ($A5$) сохраняется, имеем

$$v^M((c)_b^M) = c.$$

Получим, что $v^M((c)_b^M) = c = v^M(a)$ и по аксиоме ($A6$) имеем

$$(c)_b^M = a.$$

Теперь из $v^M(a) = c$ согласно равенству имеем

$$v^M((v^M(a))_b^M) = v^M((c)_b^M),$$

и согласно ($A6$),

$$(v^M(a))_b^M = (c)_b^M.$$

Следовательно, $(v^M(a))_b^M = (c)_b^M = a$ и получаем ($A6'$) для произвольного a.

Теперь предположим, что F имеет модель M в T_1. Тогда все аксиомы, кроме ($A6$) содержатся в M. Следовательно, для любых a и b таких, что $v^M(a) = v^M(b)$, по аксиоме ($A6'$) имеем

$$(v^M(a))_b^M = a \text{ и } (v^M(b))_b^M = b.$$

Из $v^M(a) = v^M(b)$ получим $(v^M(a))_b^M = (v^M(b))_b^M$ согласно приведенному выше равенству. Таким образом, и ($A6$) сохраняется для любых a и b. □

4. Процедура инстанцирования

Зададим процедуру трансляции формулы F в теории BLIA в эквивалентную формулу F^* в теории QF_UFLIA для того, чтобы проверить выполнимость формулы F.

Эта процедура использует аксиомы теории T_1 для составления правил инстанцирования по формуле F следующим образом:

- $(A1)$ инстанцирована с a и b для любого терма $a+_b b$ в формуле F.
- $(A2)$ инстанцирована с c и a для любого терма $c \times_b a$ в формуле F.
- $(A3)$ инстанцирована с a и b для любого терма $a \leq_b b$ в формуле F.
- $(A4)$ инстанцирована с a для любого терма типа $\mathbb{Z}_b$ в формуле F.
- $(A5)$ инстанцирована с c для любого терма вида $(c)_b$ в формуле F.
- $(A6')$ инстанцирована с a для любого терма типа $\mathbb{Z}_b$ в формуле F.

Обозначим за F^* формулу F после выполнения алгоритма инстанцирования аксиом. Обозначим за F^+ множество формул, полученных инстанцированием аксиомы $(A1)$ для любого терма $a+_b b$ в формуле F.

Аналогично определим множества $F^\times$ для $(A2)$, $F^\leq$ для $(A3)$, $F^\in$ для $(A4)$, F^c для $(A5)$, F^m для $(A6')$.

После всех трансляций, символы умножения и суммирования воспринимаются как неинтерпретируемые символы. И символом $\wedge\wedge$ обозначим операцию взятия конъюнкции всех формул из множества.

Тогда полученная формула равна следующему

$$F^* = F \wedge (\wedge\wedge F^+) \wedge (\wedge\wedge F^\times) \wedge (\wedge\wedge F^\leq) \wedge (\wedge\wedge F^\in) \wedge (\wedge\wedge F^\mathrm{c}) \wedge (\wedge\wedge F^\mathrm{m}).$$

Формула F^* решается в теории QF_UFLIA с помощью SMT-решателя. В этом подходе мы пользуемся тем, что нам не требуется разрабатывать инструмент для решения нашей задачи, вместо этого мы можем без изменения SMT-решателей, которые предназначены для широкого класса задач, произвести проверку выражений и осуществить решающую процедуру, описывающую некоторые операции и типы языка программирования Си.

$$\sum = \{+_b, \times_b, \leq_b\}, a, b \in \mathbb{Z}_b, v(a) \in \mathbb{Z}, c \in \mathbb{Z}, c - \text{константа}$$

$$\forall a, b \in \mathbb{Z}_b.\ L \leq v(a) + v(b) \leq U \Longrightarrow v(a+_b b) = v(a) + v(b), \quad (A1)$$

$$\forall a \in \mathbb{Z}_b.(c \times_b a) = c \times v(a), \quad (A2)$$

$$\forall a, b \in \mathbb{Z}_b\ a \leq_b b \Longrightarrow v(a) \leq v(b), \quad (A3)$$

$$\forall a \in \mathbb{Z}_b.\ L \leq v(a) \leq U, \quad (A4)$$

$$\forall c \in \mathbb{Z}\ .\ L \leq c \leq U \Longrightarrow v((c)_b) = c \quad (A5)$$

$$\forall a, b \in \mathbb{Z}_b.(a) + v(b) \Longrightarrow a = b \quad (A6)$$

$$\forall a \in \mathbb{Z}_b.\ ((v(a))_b = a \quad (A6')$$

Рис. 1. Аксиомы, определяющие теорию ограниченных целых
Fig. 1. Axioms defining the theory of bounded integes

2.1 Пример процедуры трансляции

Пример. Пусть

$$F = (25)_b \leq_b a \wedge -1 \times_b a +_b (25)_b \neq (0)_b.$$

И ограничения равны следующим значениям: $L = -25, U = 25$.

Шаги инстанцирования:

1.Используя аксиому $(A1)$:

$$\underline{-1 \times_b a} +_b \underline{(25)_b} \in F \Rightarrow$$

$$-25 \leq v(-1 \times_b a) + v((25)_b) \leq 25 \Rightarrow$$

$v(-1 \times_b a +_b (25)_b) = v(-1 \times_b a) + v((25)_b)$,

2.Используя аксиому (A2):

$\underline{-1} \times_b \underline{a} \in F \Rightarrow$

$-25 \leq -1 \times v(a) \leq 25 \Rightarrow v(-1 \times_b a) = -1 \times v(a)$,

3. Используя аксиому (A3):

$\underline{(25)_b} \leq_b \underline{a} \in F \Rightarrow (25)_b \leq_b a \Leftrightarrow v((25)_b)) \leq v(a)$,

4. Используя аксиому (A4):

$a \in F \Rightarrow -25 \leq v(a) \leq 25$,

5. Используя аксиому (A5):

$\underline{(25)_b} \in F \Rightarrow -25 \leq 25 \leq 25 \Rightarrow v((25)_b) = 25$,

6. Используя аксиому (A6'):

$-1 \times_b a +_b (25)_b \in F \Rightarrow (v(-1 \times_b a +_b (25)_b))_b = -1 \times_b a +_b (25)_b$.

Тогда формула F^* равна $F \wedge (-25 \leq v(-1 \times_b a) + v((25)_b) \leq 25 \Rightarrow \cdots) \wedge \ldots$. Стоит заметить, что правила A4 и A6' выглядят одинаково, но в примере были применены для разных термов, данная запись была сделана для наглядности, в программной реализации в Isabelle/HOL такие правила инстанцировния применяются для одинаковых термов. Покажем, что формула F невыполнима в логике BLIA.

Утверждение 1. *Формула F невыполнима в логике BLIA.*

Доказательство. Воспользуемся полученными формулами после применения процедуры инстанцирования. Из инстанцирования (A5) получим $v((25)_b) = 25$. Тогда из инстанцирования аксиомы (A3) получим $25 = v((25)_b)) \leq v(a)$ и из инстанцирования аксиомы (A4) имеем $v(a) \leq 25$. Таким образом, $v(a) = 25$.

Теперь из инстанцирования (A2) получим $v(-1 \times_b a) = -1 \times v(a) = -25$, тогда из (A1) $v(-1 \times_b a +_b (25)_b) = v(-1 \times_b a) + v((25)_b) = -25 + 25 = 0$.

Из аксиомы (A6') получим $-1 \times_b a +_b (25)_b = (v(-1 \times_b a +_b (25)_b))_b = (0)_b$.

Отсюда получаем следующее противоречие со вторым конъюнктом формулы F $-1 \times_b a +_b (25)_b \neq (0)_b$

Следовательно, формула F невыполнима.□

Нам нужно доказать полноту и корректность процедуры инстанцирования для теории ограниченных целых.

Корректность очевидно следует из того, что если формула F выполнима в BLIA, то формула F^* также выполнима в QF_UFLIA, так как состоит из результата инстанцирования аксиом BLIA и формулы F.

3. Доказательство полноты

Здесь мы рассмотрим произвольную формулу без свободных переменных F в теории BLIA, её трансляцию F^*, полученную через процедуру инстанцирования аксиом, и модель R, полученную из трансляции F^* в QF_UFLIA, которую мы называем реализацией.

Термы и предикаты формул F и F^* будем обозначать как $v(u) + v(t)$, а их соответствующие выражения в реализации как $v^R(u^R) +_b^R v^R(t^R)$.

Свободные термы ограниченных целых будем обозначать буквами t и u, свободные термы целых чисел обозначим через k и n, а произвольные выбранные ограниченные целые как a и b, соответствующие неограниченные целые через x, y или c.

Доказательство осуществляется с помощью восстановления модели M исходной формулы F в теории BLIA из реализации R.

Для начала докажем несколько вспомогательных лемм.

Лемма 1. *Терм вида $v(t)$ принадлежит формуле F^* тогда и только тогда, когда ограниченный целый терм t принадлежит F.*

Доказательство. Пусть терм t содержится в F. Тогда согласно правилу инстанцирования аксиомы (A4) терм $v(t)$ содержится в F^*.

Теперь предположим, что $v(t)$ принадлежит множеству термов F^*. Покажем, что этот терм возникает только благодаря процедуре инстанцирования с термом t, который принадлежит множеству термов F.

Доказательство перечислением правил инстанцирования. Рассмотрим правило (A1). Оно содержит 3 случая применения функции v, обозначаемые $v(t+_b u)$, $v(t)$ и $v(u)$.

Аксиома (A1) применяется только тогда, когда термы $t+_b u$ содержатся F. Следовательно, термы t, u и $t+_b u$ уже содержатся в F.

Доказательство относительно других правил аналогично. Случай когда $v(t)$ принадлежит множеству термов F очевиден.□

Лемма 2. *Терм вида $(v(t))_b$ принадлежит формуле F^* тогда и только тогда, когда ограниченный целый терм t принадлежит F.*

Доказательство. Пусть терм t содержится в F. Тогда согласно правилу инстанцирования аксиомы (A6') терм $(v(t))_b$ содержится в F^*.

Теперь предположим, что $(v(t))_b$ принадлежит множеству термов F^*. Аксиома (A6') применяется только тогда, когда терм t содержится в F. Случай $(v(t))_b$ принадлежит множеству термов F очевиден.□

Лемма 3. *Терм вида $(n)_b$ принадлежит формуле F^* тогда и только тогда, когда он уже принадлежит F, либо n равен $v(t)$, где t принадлежит F.*

Доказательство. Пусть терм t содержится в F. Тогда согласно лемме 2 терм $(v(t))_b$ содержится в F^*. Таким образом, $(n)_b$ принадлежит формуле F*, где $n = v(t)$.

Теперь предположим, что терм $(n)_b$ содержится в F^* и не содержится в F. В таком случае нам нужно показать, что либо $(n)_b$ содержится в F, либо n вида $v(t)$, где t содержится в F. Докажем это перечислением правил инстанцирования. Только аксиомы (A5) и (A6') содержат символ $(\cdot)_b$. Согласно правилу (A5) $(n)_b$ содержится в F, в случае (A6') n имеет требуемый вид. □

Далее дадим несколько дополнительных обозначений, которые мы будем применять в доказательствах.

Определим образ подмножества X на множестве A с помощью следующей функции $f: A \to B$ как $f[X]$. Введем следующее обозначение:

$\{f(x) \mid P(x)\} \equiv f[\{x \mid P(x)\}]$, где $\{x \mid P(x)\}$ − множество термов x, удовлетворяющих предикату $P(x)$.

Далее определим следующие множества:

$B^R \equiv \{t^R \mid v(t) \in F^*\}$,

$C^R \equiv \{n^R \mid (n)_b \in F^*\}$, где, как мы определили в начале разд. 3, t^R и n^R обозначают означивание определенных термов t и n в реализации R. Реализация R содержит частичные модели функций $v(\cdot)$ и $(\cdot)_b$ на соответствующих множества, определяемых термами в F^*. Рассмотрим функции $v^R(\cdot)$ и $(\cdot)_b^R$ (которые будем считать реализациями соответствующих функций $v(\cdot)$ и $(\cdot)_b$), определенные на множествах B^R и C^R соответственно.

Далее докажем некоторые свойства функций $v(\cdot)$ и $(\cdot)_b$.

Лемма 4. $v^R[B^R] \subseteq C^R$.

Доказательство.

$$\begin{aligned}
v^R[B^R] &= \{v^R(a) \mid a \in B^R\} \\
&= \{v^R(a) \mid a \in \{t^R \mid v(t) \in F^*\}\} \\
&= \{v^R(t^R) \mid v(t) \in F^*\} \\
&= \{v^R(t^R) \mid t \in F\} \text{ (по лемме 1)} \\
&= \{v^R(t^R) \mid (v(t))_b \in F^*\} \text{ (по лемме 2)} \\
&= \{(v(t))^R \mid (v(t))_b \in F^*\} \\
&= \{n^R \mid (n)_b \in F^* \wedge \exists t.n = v(t)\} \\
&\subseteq \{n \mid (n)_b \in F^*\} \\
&= C^R.\square
\end{aligned}$$

Лемма 5. $v^R[B^R] \subseteq [L, U]$.

Доказательство. По определению B^R для любого целого с такого, что $c \in v^R[B^R]$ имеем $c = v^R(t^R)$, где $v(t) \in F^*$. Из леммы 1 следует $t \in F$. Это, в свою очередь, означает, что аксиома (A4) была инстанцирована с термом t, то есть $L \le v(t) \le U$ − подтерм без свободных переменных в F^*. Поскольку R является моделью F^*, $L \le v^R(t^R) \le U$ и, следовательно, $L \le c \le U$ для любого $c \in B^R$.□

Лемма 6. $v^R[B^R] \subseteq C^R \cap [L, U]$.

Доказательство. Следует напрямую из лемм 4 и 5. □

Лемма 7. *v^R инъективна на множестве определения B^R.*

Доказательство. Возьмем произвольные значения ограниченно целых x и y из множества B^R. Тогда имеем $x = t^R$, $y = u^R$, $v(t) \in F^*$ и $v(u) \in F^*$. Тогда по лемме 1 $t \in F$ и $u \in F$ и, таким образом, аксиома (A6') была инстанцирована с этими термами, сохраняющие предикаты $(v(t))_b = t$ и $(v(u))_b = t$ являются подтермами без свободных переменных в F^*. Так как R модель F^*, мы имеем $(v^R(t^R))_b^R = t^R$и $(v^R(u^R))_b^R = u^R$ и поскольку R является моделью в теории QF_UFLIA, которая включает в себя сравнение неинтерпретированных функций (таких как $(\cdot)_b$) имеем $v^R(t^R) = v^R(u^R) \Rightarrow (v^R(t^R))_b^R = (v^R(u^R))_b^R$ или, что то же самое, что $v^R(t^R) = v^R(u^R) \Rightarrow t^R = u^R$, то есть $v^R(x) = v^R(y) \Rightarrow x = y$ для любых x и y из множества B^R.□

Лемма 8. *v^R сюръективна на множестве значений $C^R \cap [L, U]$.*

Доказательство.

$$\begin{aligned}
C^R &= \{n^R \mid (n)_b \in F^*\} \text{ (по определению } C^R) \\
&= \{n^R \mid (n)_b \in F\} \cup \{(v(t))^R \mid t \in F\} \text{ (по лемме 3)} \\
&= \{n^R \mid (n)_b \in F\} \cup \{(v(t))^R \mid v(t) \in F^*\} \text{ (по лемме 1)} \\
&= \{n^R \mid (n)_b \in F\} \cup \{v^R(t^R) \mid v(t) \in F^*\} \\
&= \{n^R \mid (n)_b \in F\} \cup v[\{t^R \mid v(t) \in F^*\}] \\
&= \{n^R \mid (n)_b \in F\} \cup v[B^R] \text{ (по определению} B^R).
\end{aligned}$$

Таким образом, чтобы доказать $C^R \cap [L, U] \subseteq v^R[B^R]$ мы должны показать, что $\{n^R \mid (n)_b \in F\} \cap [L, U] \subseteq v^R[B^R]$. Рассмотрим любой терм n такой, что $(n)_b \in F$.

Поскольку $(n)_b \in F$, (A5) была инстанцирована с n и, следовательно, $L \le n \le U \Rightarrow v((n)_b) = n$ является подтермом без свободных переменных в F^* и, R является моделью для F^*, $L \le n^R \le U \Rightarrow v^R((n^R)_b^R) = n^R$. Так что, если $n^R \in [L, U]$, то $n^R = v^R((n^R)_b^R) \in \{v^R((n^R)_b^R) \mid (n)_b \in F\}$. Следовательно,

$$\begin{aligned}
\{n^R \mid (n)_b \in F\} \cap [L, U] &\subseteq \{v^R((n^R)_b^R) \mid (n)_b \in F\} \\
&= \{v^R(((n)_b)^R) \mid (n)_b \in F\} \\
&= \{v^R(t^R) \mid t \in F \wedge \exists n.t = (n)_b\} \subseteq \{v^R(t^R) \mid t \in F\}
\end{aligned}$$

$$= \{v^R(t^R) \mid v(t) \in F^*\} \text{ (по лемме 1)}$$
$$= v^R[\{t^R \mid v(t) \in F^*\}]$$
$$= v^R[B^R].\square$$

Лемма 9. *v^R биективна на множествах B^Rи $C^R \cap [L, U]$ и функция $(\cdot)_b^R$ является обратной к ней.*

Доказательство. Утверждение о том, что v^R является биекцией между B^R и $C^R \cap [L, U]$ следует непосредственно из лемм 7 и 8. Таким образом, v^R имеет обратную биективную функцию. Эта обратная функция может быть однозначно охарактеризована уравнением $(v^R)^{-1}(v^R(a)) = a$ для любого $a \in B^R$. Но для каждого $a \in B^R$, значение a можно представить как t^R, где $v(t) \in F^*$. По лемме 1 терм $t \in F$ и, следовательно, аксиома (A6') была инстанцирована с термом t. Таким образом, $(v^R(t^R))_b^R = t^R$ и, следовательно, $(v^R(a))_b^R = a$ для любого $a \in B^R$. Таким образом, обратная биективная функция совпадает с функцией $(\cdot)_b^R$.$\square$

Лемма 10. $(\cdot)_b^R[C^R] \subseteq B^R$**.**

Доказательство.

$$(\cdot)_b^R[C^R] = (\cdot)_b^R[\{n^R \mid (n)_b \in F^*\}] \text{ (по определению } C^R)$$
$$= (\cdot)_b^R[\{n^R \mid (n)_b \in F\} \cup \{(v(t))^R \mid t \in F\}] \text{ (по лемме 3)}$$
$$= (\cdot)_b^R[\{n^R \mid (n)_b \in F\}] \cup (\cdot)_b^R[\{(v(t))^R \mid t \in F\}] \text{ (по определению)}$$
$$= \{(n^R)_b^R \mid (n)_b \in F\} \cup \{((v(t))^R)_b^R \mid t \in F\}$$
$$= \{((n)_b)^R \mid (n)_b \in F\} \cup \{(v^R(t^R))_b^R \mid t \in F\}.$$
$$\{((n)_b)^R \mid (n)_b \in F\} = \{t^R \mid t \in F \wedge \exists n.\, t = (n)_b\}$$
$$\subseteq \{t^R \mid t \in F\}$$
$$= \{t^R \mid v(t) \in F^*\} \text{ (по лемме 1)}$$
$$= B^R \text{ (по определению } B^R).$$

Для каждого $t \in F$ (A6') был инстанцирован с t, поэтому $(v(t))_b = t$ является подтермом без свободных переменных в формуле F^*, и поскольку R является моделью F^*, $(v^R(t^R))_b^R = t^R$. Таким образом, по определению соответствующего множества,

$$\{(v^R(t^R))_b^R \mid t \in F\} = \{t^R \mid t \in F\}$$
$$= \{t^R \mid v(t) \in F^*\} = B^R \text{ (по лемме 1 и определению } B^R).$$

Используя приведенное выше представление $(\cdot)_b^R[C^R]$ мы наконец имеем требуемое выражение $(\cdot)_b^R[C^R] \subseteq B^R$.$\square$

Получим ситуацию, изображенную на рис. 2, где v^R является биекцией между B^R и $C^R \cap [L, U]$. $(\cdot)_b^R$ является её обратной функцией на B^R и определена на множестве C^R.

Без ограничения общности, рассмотрим случай, когда $[L, U] \setminus C^R \neq \emptyset$.

Если множества C^Rи $[L, U]$ совпадают, следующее доказательство по-прежнему правильно (некоторые аргументы не принимают никаких значений и их можно не рассматривать). Соответственно, также предположим $L \leq 0 \leq U$.

Теперь рассмотрим множество $[L, U] \setminus C^R$, мощность которого не превышает $U - L + 1$. Поэтому мы произвольно выбираем несколько $|[L, U] \setminus C^R|$ различных элементов из любой области, которая не пересекается с B^R, и устанавливаем биекцию $v'(\cdot)$ между этими неявными элементами и множеством $[L, U] \setminus C^R$. Обозначим результирующий набор соответствующих отдельных неявных элементов как $\mathbb{Z}'_b$ и соответствующая обратная биекция $v'(\cdot)$ как $(\cdot)'_b$. Теперь мы готовы ввести определение восстановленной модели M (из реализации R)

исходной формулы F, которая задана на рис. 3. Далее докажем необходимые свойства этого определения.

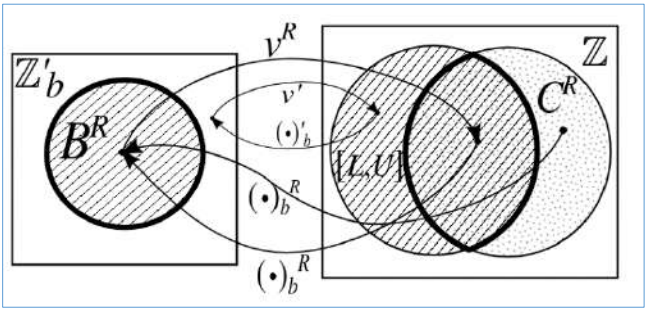

Рис. 2. Расширение биекции между множествами B^R и $C^R \cap [L,U]$ на множества $\mathbb{Z}_b^M = B^R \cup \mathbb{Z}_b{}'$ и $[L,U]$

Fig. 2. Extending the bijection between B^R and $C^R \cap [L,U]$ to the whole sets $\mathbb{Z}_b^M = B^R \cup \mathbb{Z}_b{}'$ and $[L,U]$

$$
\begin{aligned}
\mathbb{Z}_b^M &= B^R \cup \mathbb{Z}'_b,\\
v^M(a) &= \begin{cases} v^R(a), & a \in B^R,\\ v'(a), & a \notin B^R, \end{cases}\\
(c)_b^M &= \begin{cases} (c)_b^R, & c \in C^R,\\ (c)'_b, & c \in [L,U] \setminus C^R,\\ \epsilon\,\mathbb{Z}_b^M, & c \notin C^R \cup [L,U], \end{cases}\\
a +_b^M b &= \begin{cases} a +_b^R b, & (a,b) \in \left\{(t^R, u^R) \mid t +_b u \in F^*\right\},\\ \left(v^M(a) + v^M(b)\right)_b^M, & \\ \qquad (a,b) \notin \left\{(t^R, u^R) \mid t +_b u \in F^*\right\}, & \\ \qquad L \le v^M(a) + v^M(b) \le U, & \\ (0)_b, & \end{cases}\\
c \times_b^M a &= \begin{cases} c \times_b^R a, & (c,a) \in \left\{(n, t^R) \mid n \times_b t \in F^*\right\},\\ \left(c \times v^M(a)\right)_b^M, & \\ \qquad (c,a) \notin \left\{(n, t^R) \mid n \times_b t \in F^*\right\}, & \\ \qquad L \le c \times v^M(a) \le U, & \\ (0)_b, & \end{cases}\\
a \le_b^M b &= v^M(a) \le v^M(b).
\end{aligned}
$$

Рис. 3. Определение восстановленной модели M в теории BLIA

Fig. 3. Definition of the reconstructed model M in BLIA

Лемма 11. *Восстановленная модель M на рис. 3 однозначно определена*.

Доказательство. Восстановленная модель M, показанная на рис. 3, включает в себя определение области $\mathbb{Z}_b$ ограниченных целых, а также определения для всех функций из соответствующей сигнатуры теории BLIA $\{+_b, \times_b, \le_b, v(\cdot), (\cdot)_b\}$. Таким образом, мы должны показать, что функции, определенные как на рисунке, действительно отображают любую комбинацию своих аргументов, взятые из соответствующих доменов в элементы из соответствующих диапазонов. Случаи для функции v и предикат $\le_b$ очевидны. Тем не менее,

мы должны убедиться, что выбранное множество $\mathbb{Z}_b^M$ действительно содержит все значения, взятые функциями $(\cdot)_b^M$, $+_b^M$ и $\times_b^M$, иначе наше определение модели противоречиво.

Рассмотрим случаи для функции $(\cdot)_b^M$. В первом случае $(c)_b^R \in B^R$ для любого $c \in C^R$ согласно лемме 10, поэтому $(c)_b^R \in \mathbb{Z}_b^M$. Во втором случае $(c)'_b \in \mathbb{Z}'_b \subseteq \mathbb{Z}_b^M$ согласно построению.

В третьем случае $\epsilon\mathbb{Z}_b^M \in \mathbb{Z}_b^M$ по определению эпсилон-оператора Гильберта ϵ (выбор произвольного элемента непустого множества), так как $|\mathbb{Z}_b^M| \geq |\mathbb{Z}'_b| = |[L,U] \setminus C^R| > 0$ (в общем случае либо $\mathbb{Z}'_b$, либо B^R не пусты).

Теперь рассмотрим функцию $+_b^M$. Второй и третий случаи ($(v^M(a) + v^M(b))_b^M$ и $(0)_b$) явно определены, поскольку $(c)_b^M$ явно определен для любого c, как показано выше.

Во-первых, в случае, если мы поступаем по индукции по правилам инстанцирования и показываем, что терм вида $t+_b u$ может возникать только во время инстанцирования, когда он уже содержится в F (единственная аксиома (A1) инстанцируется всякий раз, когда $t+_b u$ содержится в F). Затем из $t+_b u \in F^*$ следует $t+_b u \in F$ и по лемме 1, $v(t+_b u) \in F^*$. Поскольку $a = t^R$, $b = u^R$, $a+_b b = t^R +_b^R u^R = (t+_b u)^R \in B^R \subseteq \mathbb{Z}_b^M$ по определению B^R.

Доказательство для функции $\times_b^M$ аналогично. □

Лемма 12. *Аксиомы (A4), (A5) и (A6') теории BLIA сохраняются в модели M.*

Доказательство. Рассмотрим аксиому (A4) для произвольного $a \in \mathbb{Z}_b^M = B^R \cup \mathbb{Z}'_b$. В случае $a \in B^R$, $v^M(a) = v^R(a) \in [L,U]$ по лемме 5.

В случае, если $a \in \mathbb{Z}'_b$, $v^M(a) = v'(a) \in [L,U] \setminus C^R \subseteq [L,U]$ согласно построению.

Рассмотрим аксиому (A5) для произвольного целого числа c. Согласно правилу инстанцирования аксиомы, мы должны показать равенство $v^M((c)_b^M) = c$ для любого $c \in [L,U] = (C^R \cap [L,U]) \cup ([L,U] \setminus C^R)$. В случае $c \in C^R \cap [L,U]$, используя определения на рисунке 3 и в лемме 9 мы получаем $(c)_b^M = (c)_b^R \in B^R$, а также $v^M((c)_b^M) = v^R((c)_b^R) = c$. В случае $c \in [L,U] \setminus C^R$ мы получим $(c)_b^M = (c)'_b \in \mathbb{Z}'_b$и $v^M((c)_b^M) = v'((c)'_b) = c$ согласно построению.

Наконец, в случае аксиомы (A6') мы должны показать $(v^M(a))_b^M = a$ для любого $a \in \mathbb{Z}_b^M = B^R \cup \mathbb{Z}'_b$. В случае $a \in B^R$ по лемме 9 имеем $v^M(a) = v^R(a) \in C^R \cap [L,U]$и $(v^M(a))_b^M = (v^R(a))_b^R = a$. В случае, если $a \in \mathbb{Z}'_b$, получаем $v^M(a) = v'(a) \in [L,U] \setminus C^R$ и $(v^M(a))_b^M = (v'(a))'_b = a$ согласно построению. □

Лемма 13. *Аксиомы (A1), (A2) и (A3) теории BLIA сохраняются в модели M.*

Доказательство. Сначала мы проверим аксиому (A1). Рассмотрим случаи в определении операции $+_b^M$.

Рассмотрим случай $(a,b) \in \{(t^R, u^R) \mid t+_b u \in F^*\}$. В доказательстве леммы 11 (последний абзац) показано, что из $t+_b u \in F^*$ следует $t+_b u \in F$. По правилу инстанцирования аксиомы (A1) заключаем, что $L \leq v(t) + v(u) \leq U \Rightarrow v(t+_b u) = v(t) + v(u)$является основным подтермом F^* и, поскольку R является моделью F^*, мы имеем $L \leq v^R(t^R) + v^R(u^R) \leq U \Rightarrow v^R(t^R +_b^R u^R) = v^R(t^R) + v^R(u^R)$.

Из факта $(a,b) \in \{(t^R, u^R) \mid t+_b u \in F^*\}$ следует $a = t^R$ и $b = u^R$, поэтому $L \leq v^R(a) + v^R(a) \leq U \Rightarrow v^R(a+_b^R b) = v^R(a) + v^R(b)$.

Теперь, так как $t+_b u \in F$, мы имеем $t \in F$, $u \in F$, и в силу леммы 1, $v(t+_b u) \in F^*$, $v(t) \in F^*$ и $v(u) \in F^*$. Это означает $a+_b^R b = t^R +_b^R u^R = (t+_b u)^R \in B^R$, $a = t^R \in B^R$ и $b = t^R \in B^R$ по определению B^R. Таким образом, согласно определениям на рисунке 3, $L \leq v^M(a) + v^M(b) \leq U \Rightarrow v^M(a+_b^M b) = v^M(a) + v^M(b)$.

Теперь рассмотрим случай $L \leq v^M(a) + v^M(b) \leq U$. Из леммы 12 аксиома (A5) сохраняется в M и, следовательно, $L \leq c \leq U \Rightarrow v^M((c)_b^M) = c$ для любого c и, в частности, $L \leq v^M(a) + v^M(b) \leq U \Rightarrow v^M((v^M(a) + v^M(b))_b^M) = v^M(a) + v^M(b)$.

По определению $+_b^M$ на рисунке 3 получаем $L \le v^M(a) + v^M(b) \le U \Rightarrow v^M(a +_b^M b) = v^M(a) + v^M(b)$.

Наконец, в случае $v^M(a) + v^M(b) \notin [L, U]$, предположение $L \le v^M(a) + v^M(b) \le U$ не выполняется и, следовательно, аксиома (A1) очевидно сохраняется.

Доказательство для аксиомы (A2) аналогично и в случае (A3) непосредственно следует из определения $\le_b^M$ на рис. 3□

Лемма 14. *Модель M может быть расширена неинтерпретируемыми константами, которые содержатся в F такие, что для любого подтерма $t \in F$ его интерпретации в модели M и в реализации R совпадают, то есть $t^M = t^R$.*

Доказательство. Без ограничения общности рассмотрим случай, когда t не содержит вхождения неинтерпретированных функций с арностью больше нуля. Расширение этого доказательства на термы с неинтерпретируемыми функциями (а не константами) очевидно. Доказательство проводится индукцией по структуре терма t.

Мы начнем с рассмотрения подтермов нулевой арности, то есть констант.

Интерпретируемые константы (числа) имеют одинаковые фиксированные интерпретации как в R, так и в M. Выберем интерпретацию неинтерпретируемых констант, встречающиеся в F, они будут одинаковыми в обоих моделях M и R. Это согласуется, поскольку если терм t содержится в F, то по лемме 1, $v(t)$ встречается в F^* и поэтому $t^R \in B^R \subseteq \mathbb{Z}_b^M$.

Аналогично, если t является аргументом функции v, который содержится в F и $t^M = t^R$, то $t^R \in B^R$ и по определению на рисунке 3 $v^M(t^M) = v^R(t^R)$, что по определению означивания дает $(v(t))^M = (v(t))^R$. Если $(n)_b$ входит в F, то по лемме 3 он также содержится в F^* и, таким образом, $n^R \in C^R$. Следовательно, если $n^M = n^R$ тогда $((n)_b)^M = (n^M)_b^M = (n^R)_b^M = (n^R)_b^R = ((n)_b)^R$.

Если $n \times_b t$ встречается в F, то по построению инстанцированная формула также содержится в F^*.

Если, кроме того, $t^M = t^R$, то $(n \times_b t)^M = n^M \times_b^M t^M = n^R \times_b^M t^R = n^R \times_b^R t^R = (n \times_b t)^R$. То же относится и к $+_b$.

Наконец, если $t \le u_b$ содержится в F, то согласно правилам инстанцирования, аксиома (A3) была инстанцирована с t и u, следовательно, $t^R \le_b^R u^R \Leftrightarrow v^R(t^R) \le v^R(u^R)$.

По предположению индукции имеем $v^M(t^M) = v^R(t^R)$ и $v^M(u^M) = v^R(u^R)$. Кроме того, по определению на рис. 3 у нас есть $t^M \le_b^M u^M \Leftrightarrow v^M(t^M) \le v^M(u^M)$.

Таким образом, $t^M \le_b^M u^M \Leftrightarrow v^R(t^R) \le v^R(u^R) \Leftrightarrow t^R \le_b^R u^R$.□

Теперь полнота процедуры инстанцирования напрямую следует из вышеуказанных лемм.

Теорема 2. *Каждая формула F без свободных переменных выполнима в теории BLIA тогда и только тогда, когда её трансляция F^* выполнима в QF_UFLIA.*

Доказательство. Если F выполнима в BLIA, то F^* выполнима в QF_UFLIA из-за корректности процедуры инстанцирования.

Предположим, что F^* выполнима в QF_UFLIA с моделью R. Затем по леммам 11, 12 и 13 восстановленная модель M является моделью теории BLIA. Кроме того, по лемме 14, вся формула F как терм имеет одинаковое значение в обоих R и M, расширенная соответствующими неинтерпретированными константами. Так как F является основным подтермом F^* и, следовательно, является истиной в R, она также выполнима в M. □

4. Формализация.

Доказательство полноты представлено в предыдущем разделе и было оформлено в Isabelle/HOL. В этой формализации мы использовали безтиповый синтаксис над формулами (untyped deep embedding), чтобы применять простую индукцию с двумя возможными

конструкциями (функция и переменная) в структуре формул и составить обоснование интерпретаций этих формул в различных моделях объектной логики.

Существуют два наиболее заметных различия, отличающих полностью формализованное доказательство в Isabelle/HOL от доказательств в статье.

Во-первых, использование безтипового синтаксиса над формулами (untyped deep embedding) значительно упрощает структурное представление формул, позволяет представлять бессмысленные искаженные формулы и потенциально интерпретировать их в объектной логике. Например, такие формулы: $((a)_b \leq n) + (b)_b \times_b 2$, где каждый символ функции применяется по крайней мере к одному неупорядоченному типу, который может быть представлен в нашем синтаксисе. Чтобы исключить такие формулы, мы сформулировали явное ограничение однозначной определенности (в форме специального предиката) и использовали его в качестве предварительных условий в различных леммах, а также в нашем определении выполнимости.

В нашем подходе интерпретация может только моделировать однозначно определенную формулу.

Вторая проблема не так очевидна. Традиционное определение интерпретации для формулы с кванторами (такой как аксиома) включает в себя дополнительный параметр, обычно называемый означиванием μ, который отображает неявные переменные в соответствующие им интерпретации. Таким образом, с помощью этого определения пришлось переопределять не только формулы и интерпретации (модели), но и соответствующие означивания. В этом стиле рассуждений установление казалось бы, тривиального правила подстановки, т.е. если $\forall a. F^M(a)$ мы имеем $F^M(x^M) = (F(x))^M$ если a не свободная переменная в x, требует определения промежуточного означивания $\mu' = \mu \circ [a \mapsto x^M]$ и оценивая формулы F в модели M с обоими означиваниями μ и μ'. Вместо этого мы используем определение интерпретации, основанное на подстановке, и формализуем следующим образом:

$\forall a \in \mathcal{D}. F^M(a) \Leftrightarrow (\forall t. vars(t) = \emptyset \Rightarrow ([t/a]F(a))^M)$, где любая модель M должна удовлетворять следующему ограничению:

$\forall a \in \mathcal{D}. \exists t. t^M = a$. Здесь $\mathcal{D}$ - множество, выбранное для интерпретации термов (в частности, переменных) соответствующего типа, $[t/a]$ обозначает подстановку переменной a термом t. Поскольку терм t с подстановкой не имеет свободных переменных, то нет необходимости в дополнительных методах по предотвращению замыкания переменных. Более того, в нашей формализации нам никогда не требовались вложенных кванторов, поэтому мы использовали абстрактные схематические переменные вместо кванторов и, таким образом, значительно упростили представление формулы. В самом деле, мы еще больше упростили наш подход на основе замещения путем ограничения формы целочисленных ограниченных целых термов t в подстановке.

Если $\forall a \in \mathbb{Z}_b. \exists c \in \mathbb{Z}. (c)_b = a$, мы можем перейти от кванторов над ограниченными целыми термами t к кванторам над целыми:

$(\forall t. vars(t) = \emptyset \Rightarrow ([t/a]F(a))^M) \Leftrightarrow \forall c \in \mathbb{Z}. (F(c)_b)^M$, таким образом, полностью изменяя использование кванторов в объектной логике (в BLIA/QF_UFLIA) в металогику с использованием кванторов в HOL. Это наиболее общая форма объектной логики с кванторами, которую мы используем в нашем формальном доказательстве [9] (соответствующий документ Isabelle называется TSMT_Bound_Complete).

5. Сравнение с другими тактиками

Мы реализовали решающую процедуру на основе процедуры инстанцирования, описанную в этой статье, внутри системы Isabelle в тактике расширения SMT с помощью инстанцирования кванторов триггерами.

Данная тактика называется TSMT [9] и не описана в этой работе. Здесь мы лишь кратко отметим, что это позволяет выполнить предварительное подготовку текущей цели (будучи формулой с кванторами) со всеми инстанцированиями триггеров в соответствующих подформулах в цели.

Кванторы представлены в виде лемм и триггеры подформул могут быть определены с использованием специальных атрибутов леммы.

Тактика также поддерживает корректное восстановление доказательства для формулы с инстанцированиями (используя существующие возможности тактики Z3 для доказательства в Isabelle), а также извлечение модели (контрпримера) текущей цели и показывая полученную модель для пользователя.

Оценка конкретной тактики для системы автоматизированных доказательств – это непростая задача, так как имеется не так много легкодоступных контрольных показателей для таких инструментов и подавляющее большинство доступных доказательств уже специально адаптированы для использования существующих тактик со всеми их особенностями и ограничениями. Ближайшая доступная тактика в Isabelle/HOL, которая предоставляет аналогичные возможности, это тактика uint_arith из теории HOL-Word. Она также пытается инстанцировать текущую цель дополнительными предположениями об арифметике на ограниченных целых числах и полагается на работу по упрощению цели по стандартной арифметической тактике (стандартная процедура упрощения, presburger и linarith).

Тем не менее, в отличие от нашего подхода, трансляция, реализованная в тактике uint_arith, вообще говоря, не является полной. Тактика uint_arith не очень широко используется в большинстве практических приложений, поскольку ее возможности аналогичны возможностям тактики unat_arith (похожая тактика для натуральных, а не целых чисел), которая часто используется вместо uint_arith.

Поскольку арифметика натуральных чисел не реализуется напрямую в большинстве решателей SMT, мы разрабатывали нашу тактику на арифметике целых чисел. Таким образом, чтобы оценить нашу реализацию, мы взяли несколько наиболее трудоемких целей, решаемых с помощью тактики unat_arith и вручную преобразовали их в соответствующие цели, подходящие как для uint_arith, так и для нашей тактики (преобразование в основном чисто синтаксическое, кроме добавления некоторых недостающих ограничений вида $n \geq 0$). Цели были взяты из реальных примеров с участием верифицированных функций на языке Си, таких как memcpy и quicksort, формализованные в рамках AutoCorres [10]. Пример включает в себя более 130 вызовов uint_arith, из которых мы выбрали 11 наиболее трудоемких. В добавок к существующим примерам мы добавили несколько очень простых лемм в целочисленной арифметике с ограничениями, где тактика uint_arith не может доказать цель.

Результаты оценки показаны в табл. 1. Время решения задач показывается в секундах, если оно превышает 0,1, в противном случае время помечено как < 0.1. Время для целей, не решаемых с помощью тактики, обозначается как «-» вместе с требуемым временем на попытку до возврата к результирующему состоянию доказательства (инстанцированная цель в случае uint_arith). В Isabelle/HOL процесс доказательства основан на интерактивных формальных документах, время решения важно, так как оно напрямую влияет на общее время, требуемое для обновления модели документа на каждое взаимодействие с пользователем (особенно для неструктурированных доказательств).

Табл. 1. Оценка тактики TSMT BLIA и uint_arith на формулах и реальных подзадачах из примеров AUTOCORRES.

Tabl1e 1. Evaluation of TSMT BLIA and uint_arith tactics on sample formulas and real subgoals from AUTOCORRES examples.

Цель	**Время работы, сек.**	
	uint_arith	
memcpy wp' 1	0.366	0.329

memcpy wp' 2	< 0.1	0.450
memcpy word 1	< 0.1	0.320
memcpy word 2	< 0.1	0.350
memcpy wp' 3	0.201	0.320
memcpy wp' 4	0.508	0.315
memcpy wp' 5	0.250	0.509
partition correct 1	0.468	0.581
partition correct 2	0.401	0.441
qsort unat sub sub1	< 0.1	0.526
quicksort correct 1	0.251	0.493
simple example	– (< 0.1)	0.196
simple 2a mn 1 pl b	– (1.202)	0.860
simple div2 mul2	– (0.102)	0.756
simple 2 min mn 1 pl min	– (0.897)	< 0.1
simple div3 mul3	– (0.145)	0.779

Результаты демонстрируют, что наша реализация не значительно медленнее, чем существующая тактика, несмотря на использование исчерпывающей процедуры инстанцирования, вызов внешних решателей (Z3), а также парсинг и восстановление полученных доказательств. Более того, это обеспечивает несколько заметных преимуществ.

- Тактика обладает свойством полноты, поэтому он гарантированно решит любую цель в пределах определенного класса, в то время как тактика uint_arith не полна и ее способность решить конкретную цель не так легко предсказать;
- Благодаря своей полноте, наша тактика способна не только верифицировать правильную цель, но, что еще важнее, осмысленно опровергать неверные цели в определенном классе (где тактика полна). Эта способность важна в контексте итеративной разработки доказательства, где это широко известно, что промежуточные попытки доказать неправильные утверждения более распространены, чем неудачи из-за неполноты тактики/метода доказательства.
- Наша тактика имеет примерно одинаковую производительность в случае успешной попытки доказательства и в случае поиска контрпримера, потому что оно ведет себя одинаково в обоих случаях, в то время как uint_arith последовательно пытается решить инстанцированную цель со всеми зарегистрированными арифметическими тактиками и может потратить дополнительное время на неудачные попытки.
- Наш подход легко расширяемый, так как семантика новых функциональных символов может быть легко дополнена соответствующими леммами с их триггерами. На самом деле, легкость расширяемости нашей тактики сопоставима со стандартным упрощением в Isabelle. В примерах мы расширили нашу тактику поддержкой целочисленного деления, вычитания, максимума и минимума, чтобы соответствовать возможностям стандартных арифметических тактик в Isabelle.
- Расширение нашей тактики не требует программирования на языке ML, тактика реализована поверх общей тактики TSMT (основная часть нашего инструмента инстанцирования), просто как группа лемм с соответствующими инстанцированием триггеров, в то время как uint_arith реализуется непосредственно в Isabelle/ML.

Основное ограничение нашей текущей реализации в сравнении с существующей тактикой uint_arith – невозможность правильно обрабатывать кванторы, вложенные в саму цель и не дополнены соответствующими триггерами. Наша тактика в настоящее время игнорирует любые кванторы, присутствующие в цели. Примеры, упомянутые на рисунке 4, доступны в нашем примере теории [9] (соответствующий документ Isabelle называется TSMT_Bounded_Examples).

6. Дальнейшие исследования.

Основными направлениями будущей работы включают разработку некоторых предсказуемых (хотя и неполных) подходов к обработке кванторов, встречающиеся в цели без соответствующих триггеров (стратегии для обработки кванторов внутри существующих решателей очень эффективны, но редко предсказуемы). Еще одно важное направление это формализация и оценка решения, основанного на реализации процедуры для других разрешимых теорий, таких как эффективно разрешимый фрагмент теории бит-векторов, модульная линейная целочисленная арифметика, разрешимый фрагмент теории, формализующей различные операции над списками, теория адресной арифметики с ограниченными адресами (но с Си-подобным разделением на непересекающиеся блоки памяти), теория интерпретируемых множеств (для которых есть несколько доказательств полноты, но без формализации в системе интерактивных доказательств) и прочие практические актуальные теории и их фрагменты.

7. Заключение.

В работе представлена теория ограниченных целых, позволяющая составить решающую процедуру в Isabelle/HOL на основе решателей SMT. Эта решающая процедура позволяет проверить некоторые конструкции языка программирования Си. Стоит заметить, что разработка решающих процедур по множеству различных теорий имеет важное значение при верификации программного обеспечения. Мы охарактеризовали сложность проблемы выполнимости и обеспечили эффективное сокращение сложности решения в QF_UFLIA. Наш результат показывает, что более эффективно разрабатывать специализированные алгоритмы, чем применять общий алгоритм для арифметики Пресбургера. Кроме того, в нашем случае число инстанцирований является линейным относительно длины входной формулы.

Список литературы / References

[1]. T. Nipkow, M. Wenzel, and L.C. Paulson (editors). Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Lecture Notes in Computer Science, vol. 2283, 2002, 218 p.

[2]. J. Dawson. Isabelle theories for machine words. Electronic Notes in Theoretical Computer Science, vol. 250, no. 1, 2009, pp. 55-70.

[3]. D. Babic and M. Musuvathi. Modular arithmetic decision procedure. Technical Report MSR-TR-2005-114, Microsoft Research, 2005.

[4]. N. Bjørner, A. Blass, Y. Gurevich, and M. Musuvathi. Modular difference logic is hard. Technical Report MSR-TR-2008-140, Microsoft Research, 2008.

[5]. B.-Y. Wang. On the satisfiability of modular arithmetic formulae. Lecture Notes in Computer Science, vol. 4218, 2006, pp. 186–199.

[6]. S. Böhme. Proof reconstruction for Z3 in Isabelle/HOL. In Proc. of the 7th International Workshop on Satisfiability Modulo Theories (SMT ’09), 2009.

[7]. S. Böhme and T. Weber. Fast LCF-style proof reconstruction for Z3. Lecture Notes in Computer Science, vol. 6172, 2010, pp. 179–194.

[8]. C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB Standard: Version 2.6. Department of Computer Science, The University of Iowa, Technical Report, 2017.

[9]. R. Sadykov and M. Mandrykin. Completeness of instantiation procedure for bounded linear integer arithmetic. Formalization in Isabelle/HOL. Available at: https://forge.ispras.ru/projects/tsmt/repository/, accessed April 2020.

[10]. D. Greenaway, J. Andronick, and G. Klein. Bridging the gap: Automatic verified abstraction of C. Lecture Notes in Computer Science, vol. 7406, 2012, pp. 99–115.

[11]. Kovásznai G. How Hard is Bit-Precise Reasoning? In Proc. of the 10th International Conference on Applied Informatics, 2017. pp. 179-190.

Информация об авторах / Information about authors

Рафаэль Фаритович САДЫКОВ – стажер-исследователь в ИСП РАН, аспирант кафедры МаТИС механико-математического факультета МГУ им. М.В. Ломоносова. Сфера научных интересов: верификация программ, теория распределенных вычислений, теория графов, теория автоматов, математическая логика.

Rafael Faritovich SADYKOV – intern researcher of ISP RAS, PhD student of Faculty of Mechanics and Mathematics, Moscow State University. Research interests: program verification, distributed computing theory, graph theory, automata theory, mathematical logic.

Михаил Усамович МАНДРЫКИН – младший научный сотрудник ИСП РАН, кандидат физико-математических наук по специальности «математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей». Сфера научных интересов: формальные методы верификации программ, математическая логика, функциональное программирование, системы типов в языках программирования.

Mikhail Usamovich MANDRYKIN – researcher at ISP RAS, PhD. Research interests: formal methods, program verification, mathematical logic, functional programming, type systems.

DOI: 10.15514/ISPRAS-2020-32(2)-10

On reduced forms of initialized Finite State Machines with timeouts

[1] *A.S. Tvardovskii, ORCID: 0000-0001-7705-7214 <tvardal@mail.ru>*
[2,3] *N.V. Yevtushenko, ORCID: 0000-0002-4006-1161 <evtushenko@ispras.ru>*
[1] *National Research Tomsk State University,*
36, Lenin Ave, Tomsk, 634050, Russia
[2] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[3] *National Research University Higher School of Economics,*
20, Myasnitskaya st., Moscow, 101000, Russia

Abstract. Trace models such as Finite State Machines (FSMs) are widely used in the area of analysis and synthesis of discrete event systems. FSM minimization is a well-known optimization problem which allows to reduce the number of FSM states by deriving the canonical form that is unique up to isomorphism. In particular, the latter is a base for FSM-based 'black-box' test derivation methods such as W-method and its modifications. Since nowadays the behavior of many software and hardware systems significantly depends on time aspects, classical FSMs are extended by clock variables including input and output timeouts. The minimization of a Timed Finite State Machine (TFSM) includes both state and time aspects reduction. Existing approaches allow to derive the canonical representation for non-initialized deterministic TFSMs while for initialized TFSMs, i.e., TFSMs with the designated initial state, several pair-wise non-isomorphic minimal forms can exist. In this paper, we determine initialized TFSM classes for which the minimal form is unique up to isomorphism.

Keywords: Timed Finite State Machines; minimization; timeouts; initialized TFSM.

For citation: Tvardovskii A.S., Yevtushenko N.V. On reduced forms of initialized Finite State Machines with timeouts. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 125-134. DOI: 10.15514/ISPRAS–2020–32(2)–10

Acknowledgements. This work is partly supported by RFFR Project No. 19-07-00327.

О минимизации инициальных автоматов с таймаутами

[1] *А.С. Твардовский, ORCID: 0000-0001-7705-7214 <tvardal@mail.ru>*
[2,3] *Н.В. Евтушенко, ORCID: 0000-0002-4006-1161 <evtushenko@ispras.ru>*
[1] *Национальный исследовательский Томский Государственный университет,*
634050, Россия, г. Томск, пр. Ленина, д. 36
[2] *Институт системного программирования им. В.П. Иванникова РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[3] *Национальный исследовательский университет «Высшая школа экономики»,*
101000, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. Модели с конечным числом состояний широко используются при решении задач анализа и синтеза дискретных систем, и минимизация конечных автоматов является известной проблемой оптимизации, которая позволяет уменьшить количество состояний конечного автомата,

описывающего поведение дискретной системы, на основе использования его приведенной формы. Такая форма единственна с точностью до изоморфизма для классических конечных автоматов, что, в частности, является основой для соответствующих методов синтеза тестов с гарантированной полнотой относительно «черного ящика», таких как W-метод и его модификации. Поскольку поведение современного программного и аппаратного обеспечения зачастую зависит от временных аспектов, в настоящее время классические автоматы расширяются временной переменной и связанными с ней входными и выходными таймаутами. Существующие подходы к минимизации временных автоматов охватывают оптимизацию как состояний, так и временных аспектов, и позволяют получить единственную минимальную форму для неинициальных детерминированных временных автоматов. В то же время для инициальных временных автоматов, т.е. автоматов с выделенным начальным состоянием, могут существовать различные попарно неизоморфные минимальные формы. В настоящей работе мы определяем классы инициальных временных автоматов, для которых минимальная форма единственна с точностью до изоморфизма.

Ключевые слова: временные автоматы; минимизация; таймауты; инициальные автоматы

Для цитирования: Твардовский А.С. Евтушенко Н.В. О минимизации инициальных автоматов с таймаутами. Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 125-134 (на английском языке). DOI: 10.15514/ISPRAS–2020–32(2)–10

Благодарности: Работа выполнена при частичной поддержке гранта РФФИ № 19-07-00327.

1. Introduction

Finite State Machines (FSM) [1, 2] are widely used for synthesis and analysis of discrete components of telecommunication and other hardware and software systems [3, 4, 5]. The complexity of solving many problems significantly depends on the number of states of an FSM that represents the system (component) specification; moreover, having the canonical form of a model usually simplifies solving these problems. For example, almost all FSM-based test derivation methods [6, 7] with guaranteed fault coverage for telecommunication protocols and other control systems with deterministic behavior are developed for reduced FSMs, i.e., FSMs which have different behavior at any two different states and such a reduced machine is unique for any complete deterministic FSM.

In the classical FSM theory, FSM minimization methods are well developed [1], i.e., given a deterministic complete FSM, it is well known how to derive a reduced form of the FSM that in fact is the canonical representation of a complete deterministic FSM. Nowadays time aspects become very important when describing the behavior of digital and hybrid systems, and, respectively, classical FSMs have been extended with time variables [see, for example, 8, 9, 10, 11]. A timed FSM (TFSM) is an FSM annotated with a *clock* and extended by input/output timeouts and input/output timed guards [10, 12]. Input timed guards describe the behavior at a given state for inputs which arrive during an appropriate time interval until timeout at the state expires. If no input is applied until the clock value reaches an (input) timeout then the system can spontaneously move to another state. An output timeout describes how long an applied input is processed at a given state. In the number of cases [10], an FSM with output timed guards is considered when an interval of possible output delays for processing each transition is given. In this paper, we assume that every output timeout is a non-negative integer. As a simple example, computers or mobiles can be considered when the devices move to a sleep mode if no button is pressed during an appropriate number of time units, i.e., no input is applied.

When minimizing classical FSMs (and other trace models) most attention is paid to minimizing the number of states of the machine under investigation. Differently from classical FSMs, a timed FSM can have several non-isomorphic state reduced forms and time aspects should be also taken into account when minimizing a TFSM. In [13], the notion of a time and state reduced non-initialized FSM with timed guards and timeouts is introduced. The authors also show that such minimal form is unique up to isomorphism for a non-initialized complete TFSM. However, such minimal form is not unique for an initialized complete FSM with timeouts while such machines

are widely used for modeling timed systems with a reliable reset signal. For example, a number of test derivation techniques are developed for initialized FSMs and the absence of the unique reduced form for Timed FSMs do not allow to directly apply W-methods and its derivatives to initialized TFSMs [14].

It is known that the reason for having several minimal forms is closely related to input timeouts [15], since for FSMs with timed guards there is the unique state and time reduced FSM with time guards [16]. In this paper, we determine classes of initialized FSMs with timeouts for which the unique minimal form can be derived.

The structure of the paper is as follows. Section 2 contains the preliminaries for classical and timed FSMs. In Section 3, the related work on minimizing TFSMs is briefly described. In Section 4, we determine a class of initialized TFSMs for which the minimal form is unique up to isomorphism. In Section 5, we show how the unique form can be derived for an FSM with timeouts based on its transformation to an FSM with timed guards. Section 6 concludes the paper.

2. Preliminaries

This section contains basic definitions of classical and timed Finite State Machines.

2.1 Finite State Machines

The model of a Finite State Machine (FSM) [1] is used for describing the behavior of a system that moves from state to state under input stimuli and produces predefined output responses. If the system has a reliable reset then usually the system behavior is described by an initialized FSM, i.e., by an FSM with the designated initial state. Formally, an initialized FSM is a 5-tuple $\mathsf{S} = (S, I, O, h_{\mathsf{S}}, s_0)$ where S is a finite non-empty set of states with the designated initial state s_0, I and O are input and output alphabets, and $h_{\mathsf{S}} \subseteq (S \times I \times O \times S)$ is the transition (behavior) relation. A transition (s, i, o, s') describes the situation when an input i is applied to S at the current state s and S moves to state s' and produces the output (response) o.

A trace or an *Input/Output sequence* α/γ of the FSM S at state s is a sequence of consecutive input/output pairs starting at the state s. There is a trace $\alpha/\gamma = i_1/o_1, i_2/o_2, \ldots, i_n/o_n$ at state s of FSM S if and only if there exist transitions $(s, i_1, o_1, s_1), (s_1, i_2, o_2, s_2), \ldots, (s_{n-1}, i_n, o_n, s_n)$. Given a trace α/γ, α is the *input projection* of the trace (input sequence) while γ is the corresponding *output projection* (output sequence), i.e., an output response of the FSM when the input sequence α is applied at state s. In this paper, if the converse is not explicitly stated, we consider *complete* and *deterministic* FSMs where for each state s and input sequence α there exists a single trace α/γ. Given an input sequence α of a deterministic complete FSM, state s' is the α-successor of state s in FSM S if S moves from state s to state s' when α is applied. Given an initialized FSM S, a trace α/γ is a trace of the FSM if it is a trace at the initial state of S.

2.2 Timed Finite State Machines

In this paper, a Timed FSM (TFSM) is an FSM with timeouts that can spontaneously move to another state when the timeout expires at a current state. Respectively, a TFSM is an FSM annotated with a clock (timed variable) and timeouts. A good example is a server implementation which can decline the connection when a client request is not applied within an appropriate timeout. Correspondingly, an initialized TFSM is a 6-tuple $\mathsf{S} = (S, I, O, h_{\mathsf{S}}, \Delta_{\mathsf{S}}, s_0)$ where S is a finite non-empty set of states with the designated initial state s_0, I and O are input and output alphabets, $h_{\mathsf{S}} \subseteq S \times I \times O \times S \times Z$ is the *transition relation*, Δ_{S} is the timeout function and Z is a set of *output delays* which are nonnegative integers. We consider the *timeout function* Δ_{S}: $S \to S \times (N \cup \{\infty\})$ where N is the set of positive integers: for each state this function specifies the maximum time for waiting for an input. If no input is applied until an (input) timeout expires then the system can spontaneously move to another state. By definition, for each state of the TFSM

exactly one timeout is specified. An output delay describes the number of ticks when an output has to be produced after applying an input. A transition (s, i, o, s', d) describes the situation when an input i is applied to S at the current state s. In this case, the FSM moves to state s', the clock value then is set to zero and S produces output o after d time units. Given state s of TFSM S such that $\Delta_S(s) = (s', T)$, if no input is applied before the timeout T expires then the TFSM S moves to state s' while the clock value is set to zero. If $\Delta_S(s) = (s', \infty)$ then $s' = s$; in other words, in this case, the TFSM can stay at state s infinitely long waiting for an input.

In this work, we consider *complete* and *deterministic* TFSMs where for each pair $(s, i) \in S \times I$, there exists a single transition $(s, i, o, s', d) \in h_S$.

Similar to [7], for each state s of TFSM S we consider the function $time(s, t) = s'$ that determines state s' that will be reached by the TFSM if no input is applied during t time units.

A *timed input* is a pair (i, t) where $i \in I$ and t is a real; a timed input (i, t) means that input i is applied to the TFSM at time instance t. A timed output is a pair (o, d) where $o \in O$ and d is the output delay. A sequence of timed inputs $\alpha = (i_1, t_1) \dots (i_n, t_n)$ is a *timed input sequence*, a sequence of timed outputs $\gamma = (o_1, d_1) \dots (o_n, d_n)$ is a *timed output sequence*. A sequence $\alpha/\gamma = (i_1, t_1)/(o_1, d_1) \dots (i_n, t_n)/(o_n, d_n)$ of consecutive pairs of timed inputs and timed outputs starting at the state s is a *timed I/O sequence* or a *timed trace* of TFSM S at state s. Similar to FSMs, α is an applied timed input sequence while γ is the corresponding output response of the TFSM to sequence α of applied inputs. The behavior of TFSM S at state s is the set of all timed traces at this state. For a timed input sequence α of TFSM S at state s the α-successor is defined similar to FSM.

In order to determine the output response of the TFSM at state s to a timed input (i, t), state $s' = time(s, t)$ is calculated first. State s' is a state where the TFSM moves from state s via timeout transitions such that the maximum sum Σ of all timeouts starting from state s is less than t and $t - \Sigma < T$, where $\Delta_S(s) = (s''', T)$. At the second step, a transition (s', i, o, s'', d) is used and respectively, the machine produces the output (o, d) to a timed input (i, t) applied at state s and moves to the next state s''. Thus, the output response of the TFSM to a timed input sequence at state s is iteratively determined starting from state s. Similar to FSMs, the set of all timed traces at the initial state of the initialized TFSM determines the TFSM behavior.

A timed input sequence α is a *transfer* sequence for state s or simply an *s-transfer* sequence in TFSM S if state s is the α-successor of the initial state of S. If for state s of TFSM S there exists a *transfer* timed input sequence α, then s is an *input-reachable* state, otherwise s is *input-unreachable*. By default, the initial state is input-reachable, since it is reachable by the empty input sequence. If for an input-unreachable state s' there exists a time instance t and *input-reachable* state s such that $time(s, t) = s'$ then s' is time-reachable. The TFSM S is initially *connected* or simply *connected* if each state $s \in S$ is input- or time-reachable. In this paper, we consider only connected TFSMs.

Example. Consider a TFSM S in Figure 1 with the initial state a. This TFSM is connected but state b is input-unreachable.

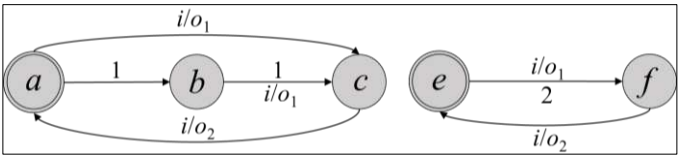

Fig. 1. State and time reduced equivalent initialized TFSMs S with the initial state a (on the left) and P with the initial state e (on the right)

Рис. 1. Приведённые по состояниям и времени инициальные временные автоматы S (слева) и P (справа) с начальными состояниями a и e соответственно

3. Minimization problem

There is a big body of work for minimizing a classical deterministic complete FSM based on the state equivalence [1]. We now remind the notion of equivalent states for deterministic complete (Timed) FSMs [1, 12]. Given complete deterministic (Timed) FSMs S and P with their states s and p, states s and p are *equivalent* if (timed) output responses at these states coincide for each (timed) input sequence. If states s and p are not equivalent then they are *distinguishable*. Initialized (Timed) FSMs are *equivalent* if their initial states are equivalent. Equivalent (Timed) FSMs have the same behavior. Two TFSMs S and P are *isomorphic* if there exists the one-to-one correspondence H: $S \to P$ such that there exists a transition $(s, i, o, s', d) \in h_S$ if and only if there exists a transition $(H(s), i, o, H(s'), d) \in h_P$ and $\Delta_P(H(s)) = (H(s'), T)$ if and only if $\Delta_S(s) = (s', T)$. A complete deterministic (Timed) FSM S is *state reduced* if every two different states $s_1, s_2 \in S$ are not equivalent.

3.1 FSM Minimization

The minimal form of a (initialized) deterministic complete FSM S is defined as a (initialized) state reduced FSM which is equivalent to S. An algorithm for deriving the minimal form (or an FSM minimization algorithm) for classical deterministic complete FSMs has been proposed in [1] and is based on partitioning the state set into equivalence classes, i.e., subsets with pairwise equivalent states. The equivalence relation induces a partition E of the set of states of a complete deterministic FSM. Any two states of the same class of the partition E are equivalent; any two states of different classes of partition E are distinguishable. Respectively, states of the reduced form correspond to classes of the partition E, i.e., the number of states of the reduced form equals the cardinality of E. The transition relation of the reduced form is derived based on transitions between E classes. It is shown that the minimal form of an initialized FSM as well as of a non-initialized deterministic complete FSM is unique up to isomorphism.

3.2 TFSM Minimization

The notions of a state reduced TFSM and the partition into equivalent states are defined similar to those of classical FSMs. A state reduced form of an FSM with timeouts can be derived based on its FSM abstraction [13, 15]. Moreover, in order to derive the unique minimal form up to isomorphism for a non-initialized FSM with timeouts, a so-called time reduced form should be also constructed.

A non-initialized FSM with timeouts S is *time reduced* if for each state s such that $\Delta_S(s) = (s', T)$, it holds that for each state $s'' \in S$ and integer $T' < T$, TFSM S′ which is obtained from S by replacing the timeout at state s to $\Delta_S(s) = (s'', T')$, is not equivalent to S. Minimal timeouts for states of TFSM S can be found based on its FSM abstraction [13]. In other words, when deriving a time reduced form of a non-initialized TFSM, the timeout for each state s should be set to the minimum value in such a way that the TFSM behavior at state s is not changed.

In [13], the following theorem has been proven.

Theorem 1. Two non-initialized deterministic complete state and time reduced FSMs with timeouts are equivalent if and only if they are isomorphic.

Respectively, the minimal form of a non-initialized deterministic complete TFSM is unique up to isomorphism.

3.3 The uniqueness of the minimal form of initialized FSMs with timeouts

Unlike non-initialized Timed FSMs, a complete deterministic initialized FSM with timeouts can have several non-isomorphic state and time reduced forms. An example is shown in Figure 1 where two equivalent minimal initialized FSMs with timeouts are presented. However, those TFSMs are not isomorphic. The reason is that for non-initialized equivalent TFSMs, for each

state of one machine there is an equivalent state of another machine and vice versa. For initialized machines it is not the case for input-unreachable states.

In fact, there can exist some states in TFSMs which are only time-reachable, i.e., are not input-reachable, for example, state *b* of TFSM S (Figure 1). This input-unreachable state can be removed from the TFSM if the TFSM moves from state *a* to state *c* via a timeout transition. The corresponding TFSM P is presented in the Figure 1 on the right where the timeout of value two is used instead of two timeouts of value one. Therefore, the problem that a reduced form of an FSM with timeouts is not unique is closely related to states which are only time-reachable because for these states, the requirement for the one-to-one correspondence between states of initialized equivalent TFSMs does not need to necessarily hold. In this paper, we specify classes of initialized FSMs with timeouts for which the state and time reduced form is unique up to isomorphism.

4. Input-connected FSM with timeouts

Since the existence of several state and time reduced equivalent but non-isomorphic initialized FSMs with timeouts is closely related to time-reachable states, we first consider TFSMs without such states.

An FSM with timeouts S is *input-connected* if each state $s \in S$ is input-reachable. In other words, given a state *s* of an input-connected initialized FSM with timeouts, there exists a transfer timed input sequence from initial state to state *s*. For input-connected TFSMs, the following proposition can be proven.

Proposition 1. Two deterministic complete state and time reduced initialized input-connected FSMs with timeouts are equivalent if and only if they are isomorphic.

Proof. Let deterministic complete state and time reduced initialized input-connected FSMs S and P with timeouts be equivalent. Since the TFSMs are equivalent then α-successors *s* and *p* of initial states TFSMs S and P respectively are equivalent for any timed input sequence α. Moreover, since S and P are state reduced then one-to-one correspondence H_i: $S \rightarrow P$ can be established such that $p = H_i(s)$ is an input-reachable state of TFSM P which is equivalent to the input-reachable state *s*. Similar to [13], for each transition $(s, i, o, s', d) \in h_S$ there exists a transition $(H_i(s), i, o, H_i(s'), d) \in h_P$. Let there exist a pair of states *s* and $p = H_i(s)$ such that $\Delta_S(s) = (s', T_s)$ and $\Delta_P(p) = (p', T_p)$ and $T_p < T_s$. Due to the fact that S and P are equivalent, there exists state s'' which is equivalent to state $p' = H_i(s'')$. Since states *s* and *p* are also equivalent, the timeout at state *s* can be replaced by $\Delta_S(s) = (s'', T_s')$ where $T_s' = T_p < T_s$. The latter is not possible as S is time reduced. Since P is time reduced too, the same reasoning can be applied when $T_s < T_p$ and thus, $T_s = T_p$. Since states *s* and *p* are equivalent, states $time(s, T_s)$ and $time(p, T_p)$ are also equivalent and respectively, $p' = H(s')$. Thus, $\Delta_P(H_i(s)) = (H_i(s'), T)$ if and only if $\Delta_S(s) = (s', T)$.

Since isomorphic TFSMs coincide up to state renaming, isomorphic TFSMs are equivalent.

Corollary. The minimal (state and time reduced) form of an input-connected initialized FSM with timeouts is unique up to isomorphism.

However, there exist FSMs with timeouts which have time-reachable states, i.e., are not input-connected. We next consider such TFSMs and discuss when the unique minimal form for TFSMs with input-unreachable states can be derived.

5. Minimization procedure using FSMs with timed guards

Given an FSM with timeouts, according to the corollary to Proposition 1, if this FSM has a state and time reduced input-connected form, then this form is unique up to isomorphism. However, just now we do not know whether such a minimal form exists for any FSM with timeouts. On the other hand, FSMs with timed guards can be considered as another type of the minimal form and such

TFSMs are input-connected. It is known [12] that there exists a class of FSMs with timeouts which can be represented by a corresponding FSM with timed guards. Respectively, in this section, we show that if for an FSM with timeouts there exists a minimal form with timed guards then such minimal form is unique up to isomorphism.

5.1 FSM with timed guards

An initialized FSM with timed guards is a 5-tuple $\mathsf{S} = (S, I, O, s_0, h_\mathsf{S})$ where I and O are input and output alphabets, S is the finite non-empty set of states with the designated initial state s_0, $h_\mathsf{S} \subseteq (S \times I \times O \times S \times \Pi \times Z)$ is the *transition relation* with the set of *input timed guards* Π. An input timed guard $g \in \Pi$ describes the time domain when a transition can be executed and is given in the form of interval $\lceil min, max \rceil$ from $[0; \infty)$, where $\lceil \in \{(, [\}, \rceil \in \{),]\}$. The transition $(s, i, o, s', g, d) \in S \times I \times O \times S \times \Pi \times Z$ means that TFSM S being at state s accepts an input i applied at time $t \in g$ measured from the moment when TFSM S entered state s; the clock then is set to zero, S produces output o and moves to state s' after d time units.

The state reduced form for FSM with timed guards can be derived based on its FSM abstraction [13] or by the algorithm presented in [16]. Moreover, an FSM with timed guards is *time reduced* if for each two transitions (s, i, o, s', g_1, d), $(s, i, o, s', g_2, d) \in h_\mathsf{S}$ it holds that timed guards g_1 and g_2 cannot be merged into a single guard. Thus, in order to derive the time reduced form for an FSM with timed guards, transitions under the same input with the same output and output delay, between the same states, the timed guards which can be merged should be replaced by a single transition. The uniqueness of the minimal form of a non-initialized FSM with timed guards has been proven in [13] and next we formulate a similar proposition for initialized TFSMs.

Proposition 2. Two deterministic complete time and state reduced initialized connected FSMs with timed guards are equivalent if and only if they are isomorphic.

Proof. Let deterministic complete time and state reduced initialized connected FSMs with timed guards S and P be equivalent. Since TFSMs S and P are equivalent, the α-successors s and p of initial states of these TFSMs are also equivalent for each α, i.e., at these states there are the same output responses for each timed input sequence. Moreover, since S and P are state reduced then the one-to-one correspondence H: $S \to P$ can be established such that $p = H(s)$ is a state of TFSM P which is equivalent to state s. We now show that for each transition $(s_1, i, o, s_2, g, d) \in \lambda_\mathsf{S}$ there exists a transition $(p_1, i, o, p_2, g, d) \in \lambda_\mathsf{P}$, where $p_1 = H(s_1)$ and $p_2 = H(s_2)$. Let there exist $(s_1, i, o, s_2, g, d) \in \lambda_\mathsf{S}$, but $(H(s_1), i, o, H(s_2), g, d) \notin \lambda_\mathsf{P}$. Since s_1 and p_1 are equivalent, the behavior of P at state p_1 in time interval g coincides with that of the TFSM S at state s_1. Respectively $(p_1, i, o, p_2, g', d) \in \lambda_\mathsf{P}$, where $g \subset g'$, because P is state and time reduced. However, the behavior of TFSM S for the same input i at state s for time instances in g differs from that in adjacent intervals since S is state and time reduced. Thus, there exists $t \in g' \setminus g$, such that the output responses to (i, t) at states s and p do not coincide. In a similar way, we can show that for each transition $(H(s_1), i, o, H(s_2), g, d) \in \lambda_\mathsf{P}$ there exists a transition $(s_1, i, o, s_2, g, d) \in \lambda_\mathsf{S}$ and thus, S and P are isomorphic.

Since isomorphic TFSMs coincide up to state renaming, isomorphic TFSMs are equivalent.

Thus, the minimal (state and time reduced) form of an initialized FSM with timed guards is unique up to isomorphism and next we show how FSM with timeouts can be represented by a TFSM with timed guards in some cases.

5.2 Transformation of an FSM with timeouts into an FSM with timed guards

An FSM with timeouts S is *timeout loop-free* [12] if there is no cycle of transitions labeled with timeouts. A timeout loop-free FSM can be represented as an FSM with timed guards by the algorithms proposed in [12].

Algorithm 1: Transforming a timeout loop-free FSM into an FSM with timed guards

Input: A timeout loop-free FSM $\mathsf{S}' = (S, I, O, h_{\mathsf{S}}, \Delta_{\mathsf{S}}, s_0)$

Output: The FSM with timed guards S

While there exists $s_j \in S$ such that $\Delta_{\mathsf{S}}(s_j) = (s_k, T)$, $T < \infty$ **do**

for each $(s_k, i, [t_1, t_2), o, s_h, d) \in h_{\mathsf{S}}$ **do** $h_{\mathsf{S}} = h_{\mathsf{S}} \cup \{(s_j, i, [t_1 + T, t_2 + T), o, s_h, d)\}$;

if $\Delta_{\mathsf{S}}(s_k) = (s_f, T_f)$ **then** $\Delta_{\mathsf{S}}(s_j) = (s_f, T_f + T)$ **else** $\Delta_{\mathsf{S}}(s_j) = (s_j, \infty)$;

Return $\mathsf{S} = (S, I, O, h_{\mathsf{S}}, s_0)$.

Proposition 3 [12]. Given a complete deterministic initialized timeout loop-free FSM S', let S be an FSM with time guards returned by Algorithm 1 for S'. TFSM S is a complete deterministic initialized FSM with timed guards that is equivalent to S'.

Let S be an FSM with timed guards that is returned by Algorithm 1 for a timeout loop-free FSM S'. Note that each time-reachable state of TFSM S' becomes unreachable from the initial state of TFSM S and, respectively, can be removed since the behavior at this state does not affect the machine behavior at the initial state. As an example, for the timeout loop-free FSM S in Figure 1 the corresponding FSM with timed guards Q is presented in Figure 2.

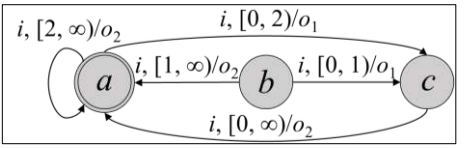

Fig. 2. An FSM with timed guards Q that is equivalent to FSM with timeouts S in Fig. 1
Рис. 2. Автомат с временными ограничениями Q, эквивалентный автомату с таймаутами S на рисунке 1

State *b* of TFSM Q becomes unreachable from the initial state *a* and can be removed without changing the initialized TFSM behavior. The following proposition can be proven based on Proposition 2 and results from [12].

Proposition 4. Two deterministic complete initialized timeout loop-free FSMs are equivalent if and only if corresponding state and time reduced FSMs with timed guards are isomorphic.

Given an initialized timeout loop-free FSM S, the state and time reduced form can be derived for S as such a form for a corresponding FSM with timed guards. Respectively, the unique minimal form for FSM with timed guards or timeouts P can be derived by the following algorithm.

Algorithm 2: Deriving the minimal form of an initialized FSM with timed guards or timeouts

Input: An initialized FSM with timed guards or an initialized FSM with timeouts S

Output: The minimal FSM with timed guards S'

Step 1. If S is an FSM with timeouts then Step 2, else Step 3.

Step 2. If S is a timeout loop-free FSM then call Algorithm 1 for deriving a corresponding FSM with timed guards and Step 3, else the unique minimal form as an FSM with timed guards cannot be derived.

Step 3. The unique minimal form with timed guards S' is derived as described in [16].

Proposition 5. Let P be a TFSM returned by the Algorithm 2 for a complete deterministic initialized timeout loop-free FSM or an FSM with timed guards S. TFSM P is a complete deterministic initialized state and timed reduced FSM with timed guards which is equivalent to S.

Given an FSM with timed guards or timeouts, we next determine a corresponding class of TFSMs with the unique minimal form in the following way.

Theorem 2. Given two deterministic complete FSMs with timed guards or timeouts S and P which are initialized connected timeout loop-free TFSMs, S and P are equivalent if and only if their time and state reduced forms of corresponding FSMs with timed guards are isomorphic.

6. Conclusion

In this work, we have investigated the uniqueness of the minimal form for initialized FSMs with timeouts. We determine two TFSM classes for which the minimal form is unique up to isomorphism. The first class contains initialized TFSMs for which all states are reachable from initial state under a timed input sequence. The second class contains TFSMs which are timeout loop-free, i.e., their transition diagrams have no loops labeled with timeouts.

We also note that the uniqueness of the minimal form for Timed FSMs allows to directly adapt classical W-based test derivation methods for TFSMs. These methods are based on checking the equivalence relation by checking the isomorphism (or possibly another relation for nondeterministic TFSMs) between the specification and an implementation under test. We also plan to study the possibility of deriving homing and synchronizing sequences for FSMs with timeouts based on their minimal forms.

References / Список литературы

[1]. A. Gill. Introduction to the Theory of Finite-State Machines, McGraw Hill, 1962, 207 p.
[2]. D. Lee, M. Yannakakis Principles and methods of testing finite state machines-a survey. Proceedings of the IEEE, vol. 84, no. 8, 1996, pp. 1090-1123.
[3]. T.E. Murphy, X.-J. Geng, and J. Hammer. On the control of asynchronous machines with races. IEEE Transactions on Automatic Control, vol. 48, no. 6, 2003, pp. 1073-1081.
[4]. Kumar Ratnesh, Garg Vijay K. Modeling and control of logical discrete event systems. The Springer International Series in Engineering and Computer Science, 1995, 143 p.
[5]. C. C. Cassandras, S. Lafortune. Introduction to discrete event systems. Springer, 1999, 828 p.
[6]. Rita Dorofeeva et al. FSM-based conformance testing methods: A survey an-notated with experimental evaluation. Information and Software Technology, vol. 52, issue 12, 2010, pp. 1286-1297.
[7]. Maxim Zhigulin, Nina Yevtushenko, Stéphane Maag, Ana R. Cavalli. FSM-Based Test Derivation Strategies for Systems with Time-Outs. In Proc. of the 11th International Conference on Quality Software, 2011, pp. 141-149.
[8]. Tripakis S. Folk theorems on the determinization and minimization of timed automata. Information Processing Letters, vol. 99, no. 6, 2006, pp. 222-226.
[9]. Merayo M., Nunez M., Rodriguez I. Formal testing from timed finite state machines, Computer Networks, vol. 52, issue 2, 2008, pp. 432-460.
[10]. Gromov M., El-Fakih K., Shabaldina N., and Yevtushenko N. Distinguishing non-deterministic timed fnite state machines. In Formal Techniques for Distributed Systems. Lecture Notes in Computer Science. vol. 5522, 2009, pp.137-151.
[11]. R.M. Hierons, M.G. Merayo, M. Nunez. Testing from a Stochastic Timed System with a Fault Model / // Journal of Logic and Algebraic Programming, vol. 72, issue 8, 2009, pp. 98-115.
[12]. Bresolin D., El-Fakih K., Villa T., Yevtushenko N. Deterministic Timed Finite State Machines: Equivalence Checking and Expressive Power. In Proc. of the 5th International Symposium on Games, Automata, Logics and Formal Verification (GandALF2014), 2014, pp. 203–216.
[13]. Tvardovskii A.S., Yevtushenko N.V., Gromov M.L. Minimizing Finite State Machines with time guards and timeouts. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017 г., pp. 139-154 (in Russian). DOI: 10.15514/ISPRAS-2017-29(4)-8 / Твардовский А.С., Евтушенко Н.В., Громов М.Л. Минимизация автоматов с таймаутами и временными ограничениями. Труды ИСП РАН, том 29, вып. 4, 2017 г., стр. 139-154.
[14]. Khaled El-Fakih, Nina Yevtushenko, Hacène Fouchal. Testing Timed Finite State Machines with Guaranteed Fault Coverage. Lecture Notes in Computer Science, vol. 5826, 2009. pp. 66-80.
[15]. Tvardovskiy A. On the minimization of timed Finite State Machines. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 6, 2014, pp. 77-84 (in Russian). DOI: 10.15514/ISPRAS-2014-26(6)-7 / Твардовский А.С. К минимизации автоматов с таймаутами, том 26, вып. 6, 2014 г., стр. 77-84.
[16]. Tvardovskiy A.S., Yevtushenko N.V. Minimizing Timed Finite State Machines. Tomsk State University Journal of Control and Computer Science, № 4(29), 2014, pp. 77-83 (in Russian) / Твардовский А.С., Евтушенко Н.В. К минимизации автоматов с временными ограничениями. Вестник Томского государственного университета. Управление, вычислительная техника и информатика, 2014 г., № 4(29), стр. 77–83.

Information about authors / Информация об авторах

Aleksander Sergeevitch TVARDOVSKII received his Master's degree from Faculty of Radiophysics of Tomsk State University. He is a Ph.D. student at Tomsk State University since 2017. His research interests include automata theory and software testing.

Александр Сергеевич ТВАРДОВСКИЙ получил степень магистра радиофизики в ТГУ. С 2017 года обучается в аспирантуре ТГУ. Исследовательские интересы включают теорию автоматов и тестирование программного обеспечения.

Nina Vladimirovna YEVTUSHENKO, doctor of technical sciences, professor, a leading researcher of ISP RAS, worked at the Siberian Scientific Institute of Physics and Technology as a researcher up to 1991. In 1991, she joined Tomsk State University as a professor and then worked as the chair head and the head of Computer Science laboratory. Her research interests include formal methods, automata theory, distributed systems, protocol and software testing.

Нина Владимировна ЕВТУШЕНКО, доктор технических наук, профессор, г.н.с. ИСП РАН, до 1991 года работала научным сотрудником в Сибирском физико-техническом институте. С 1991 г. работала в ТГУ профессором, зав. кафедрой, зав. лабораторией по компьютерным наукам. Её исследовательские интересы включают формальные методы, теорию автоматов, распределенные системы, протоколы и тестирование программного обеспечения.

DOI: 10.15514/ISPRAS-2020-32(2)-11

Модификация алгоритма Валианта для задачи поиска подстрок

Ю.А. Сусанина, ORCID: 0000-0003-3904-3764 <jsusanina@gmail.com>
А.Н. Явейн, ORCID: 0000-0001-9733-5429 <yaveyn@yandex.ru>
С.В. Григорьев, ORCID: 0000-0002-7966-0698 <s.v.grigoriev@spbu.ru>

Санкт-Петербургский государственный университет,
199034, Россия, г. Санкт-Петербург, Университетская наб., д. 7/9

Аннотация. Теория формальных языков активно изучается и находит широкое применение во многих областях. Например, в биоинформатике в задачах распознавания и классификации иногда требуется найти подпоследовательности генетических цепочек, обладающие некоторыми характерными чертами, которые могут быть описаны с помощью грамматики. Задача поиска этих подпоследовательностей сводится к проверке их принадлежности некоторому формальному языку, заданному грамматикой. При этом часто требуется эффективная обработка больших объёмов данных, что приводит к необходимости усовершенствования существующих методов синтаксического анализа. На данный момент среди алгоритмов синтаксического анализа, работающих с произвольной КС-грамматикой, одним из самых быстрых является алгоритм Валианта, основанный на использовании матричных операций. В данной работе предложен алгоритм, который является модификацией алгоритма Валианта. Его основным достоинством является возможность разбиения матрицы разбора на подслои непересекающихся подматриц, которые могут быть обработаны независимо. Доказана корректность и приведена оценка сложности предложенного алгоритма. Проведенные эксперименты показывают, что он сохранил основные преимущества исходного алгоритма, главное из которых – высокая производительность, полученная за счет использования эффективных методов перемножения матриц. Также предложенный алгоритм позволил заметно уменьшить время, затрачиваемое на поиск подстрок, сократив большое количество избыточных вычислений.

Ключевые слова: синтаксический анализ; контекстно-свободные грамматики; матричные операции

Для цитирования: Сусанина Ю.А., Явейн А.Н., Григорьев С.В. Модификация алгоритма Валианта для задачи поиска подстрок. Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 135-148. DOI: 10.15514/ISPRAS–2020–32(2)–11

Благодарности: Данная работа выполнена при поддержке гранта РНФ 18-11-00100 и JetBrains Research.

Modification of Valiant's algorithm for the string-matching problem

Y.A. Susanina, ORCID: 0000-0003-3904-3764 <jsusanina@gmail.com>
A.N. Yaveyn, ORCID: 0000-0001-9733-5429 <yaveyn@yandex.ru>
S.V. Grigorev, ORCID: 0000-0002-7966-0698 <s.v.grigoriev@spbu.ru>
Saint Petersburg State University,
7/9, Universitetskaya nab., St. Petersburg, 199034, Russia

Abstract. The theory of formal languages and, particularly, context-free grammars has been extensively studied and applied in different areas. For example, several approaches to the recognition and classification problems in bioinformatics are based on searching the genomic subsequences possessing some specific features which can be described by a context-free grammar. Therefore, the string-matching problem can be reduced to parsing

– verification if some subsequence can be derived in this grammar. Such field of application as bioinformatics requires working with a large amount of data, so it is necessary to improve the existing parsing techniques. The most asymptotically efficient parsing algorithm that can be applied to any context-free grammar is a matrix-based algorithm proposed by Valiant. This paper aims to present Valiant's algorithm modification, which main advantage is the possibility to divide the parsing table into successively computed layers of disjoint submatrices where each submatrix of the layer can be processed independently. Moreover, our approach is easily adapted for the string-matching problem. Our evaluation shows that the proposed modification retains all benefits of Valiant's algorithm, especially its high performance achieved by using fast matrix multiplication methods. Also, the modified version decreases a large amount of excessive computations and accelerates the substrings searching.

Keywords: parsing, context-free grammars, matrix operations.

For citation: Susanina Y.A., Yaveyn A.N., Grigorev S.V. Modification of Valiant's algorithm for the string matching problem. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 135-148 (in Russian). DOI: 10.15514/ISPRAS–2020–32(2)–11

Acknowledgements. The research was supported by the Russian Science Foundation grant 18-11-00100 and a grant from JetBrains Research.

1. Введение

Теория формальных языков активно изучается и находит широкое применение во многих областях [2], прежде всего, в информатике, для описания языков программирования. Также существует множество исследований, которые показывают эффективность использования КС-грамматик в биоинформатике [12, 13]. Так, формальные языки применяются для решения задач распознавания и классификации в биоинформатике, некоторые из которых основаны на том, что вторичная структура последовательностей ДНК и РНК содержит в себе важную информацию об организме [16]. Характерные особенности вторичной структуры могут быть описаны с помощью КС-грамматики [14, 15], что позволяет свести проблему распознавания и классификации к задаче синтаксического анализа (определения принадлежности некоторой строки к языку, заданному грамматикой).

Большинство алгоритмов синтаксического анализа либо работают за кубическое время (Касами [3], Янгер [4], Эрли [10]), либо применяются только к определенным подклассам КС-грамматик (Бернарди, Клауссен [11]). На данный момент одним из самых быстрых алгоритмов, работающих с любой КС-грамматикой, является алгоритм Валианта [5]. Более того, данный алгоритм был расширен для конъюнктивных и булевых грамматик, которые обладают большей выразительностью, чем КС-грамматики [1,7,8]. Однако алгоритм Валианта плохо применим к проблеме нахождения всех подстрок определенной длины, так как он будет выполнять много лишних вызовов перемножения матриц.

В данной работе предложен алгоритм, который является модификацией алгоритма Валианта. За счет изменения порядка вычисления перемножений матриц появилась возможность разбиения матрицы разбора на слои непересекающихся подматриц. Предложенный подход частично решает проблему поиска подстрок за счет простой остановки алгоритма после заполнения определенного слоя. Кроме того, каждая матрица слоя может обрабатываться независимо, что в дальнейшем позволит повысить эффективность алгоритма, используя техники параллельных вычислений. Доказана корректность предложенного алгоритма и приведена оценка сложности. Проведены эксперименты, показывающие, что предложенный алгоритм не проигрывает в производительности алгоритму Валианта и может быть эффективно применен к задаче поиска подстрок.

Работа организована следующим образом. В разд. 2 даны основные понятия и приведен исходный алгоритм Валианта; в разд. 3 представлен алгоритм, являющийся модификацией алгоритма Валианта, легко адаптируемый к задаче поиска подстрок и позволяющий повысить использование параллельных техник, а также доказана корректность и приведена оценка

сложности модифицированный версии; в разд. 4 показана применимость предложенного нами алгоритма к задаче поиска подстрок; в разд. 5 представлены результаты проведенных экспериментов; заключение и направления будущих исследований приведены в разделе 6.

2. Обзор

В этом разделе мы введем основные определения и опишем алгоритм Валианта, на котором основывается предложенная в данной работе модификация.

2.1 Терминология

Грамматикой будем называть четверку (Σ, N, R, S), где Σ – конечное множество терминальных символов, N – конечное множество нетерминальных символов, R – конечное множество правил вида $\alpha \to \gamma$, где $\alpha \in V^*NV^*, \gamma \in V^*, V = \Sigma \cup N$ и $S \in N$ – стартовый символ.

Грамматика называется контекстно-свободной (КС), если любое ее правило $r \in R$ имеет вид $A \to \beta$, где $A \in N, \beta \in V^*$.

КС-грамматика $G = (\Sigma, N, R, S)$ называется грамматикой в нормальной форме Хомского, если любое ее правило имеет одну из следующих форм:

- $A \to BC$,
- $A \to a$,
- $S \to \varepsilon$ (если пустая строка $\varepsilon \in L_G$),

где $A, B, C \in N$, $a \in \Sigma$, L_G – язык, порождаемый грамматикой G.

С помощью $L_G(A)$ будем обозначать язык, порождаемый грамматикой $G_A = (\Sigma, N, R, A)$.

2.2 Алгоритм Валианта

Задачей синтаксического анализа является проверка принадлежности входной строки языку, порождаемому некоторой грамматикой.

Взятый за основу в данной статье алгоритм Валианта относится к табличным методам синтаксического анализа, главная идея которых – построение для входной строки $a = a_1 \dots a_n$ и КС-грамматики в нормальной форме Хомского $G = (\Sigma, N, R, S)$ таблицы (далее – матрицы) разбора T размера $(n+1) \times (n+1)$, где $T_{i,j} = \{A\colon A \in N, a_{i+1} \dots a_j \in L_G(A)\}$ для всех $i < j$.

Элементы матрицы T должны заполняться последовательно, начиная с диагонали: $T_{i-1,i} = \{A\colon A \to a_i \in R\}$.

Затем $T_{i,j}$ вычисляются по формуле $T_{i,j} = f(P_{i,j})$, где

$$P_{i,j} = \bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j},$$

$$f(P_{i,j}) = \{A | \exists A \to BC \in R, (B, C) \in P_{i,j}\}.$$

Входная строка $a = a_1 \dots a_n$ принадлежит языку L_G тогда и только тогда, когда $S \in T_{0,n}$.

Если все элементы матрицы T заполнять последовательно, то вычислительная сложность данного алгоритма будет равна $O(n^3)$. Наиболее затратной по времени операцией является вычисление $\bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$. Валиант предложил реорганизовать порядок заполнения элементов матрицы разбора так, что стало возможным выполнить эти вычисления как перемножение некоторого количества булевых матриц.

Входные данные: G – КС-грамматика, $a = a_1 \dots a_n$, $a_i \in \Sigma, n \geq 1$, где $n+1$ –степень двойки
main():
 $compute\ (0, n+1)$;
 accept if and only if $S \in T_{0,n}$
$compute\ (l, m)$:
 if $m - l \geq 4$ **then**
  $compute(l, \frac{l+m}{2})$;
  $compute(\frac{l+m}{2}, m)$
 $complete\ (l, \frac{l+m}{2}, \frac{l+m}{2}, m)$
$complete\ (l, m, l', m')$:
 if $m - l = 1$ **and** $m = l'$ **then**
  $T_{i-1,i} = \{A: A \to a_i \in R\}$;
 else if $m - l = 1$ **and** $m < l'$ **then**
  $T_{i,j} = f(P_{i,j})$;
 else if $m - l > 1$ **then**
  $leftgrounded = \left(l, \frac{l+m}{2}, \frac{l+m}{2}, m\right)$; $rightgrounded = \left(l', \frac{l'+m'}{2}, \frac{l'+m'}{2}, m'\right)$;
  $bottom = (\frac{l+m}{2}, m, l', \frac{l'+m'}{2})$; $left = (l, \frac{l+m}{2}, l', \frac{l'+m'}{2})$;
  $right = (\frac{l+m}{2}, m, \frac{l'+m'}{2}, m')$; $top = (l, \frac{l+m}{2}, \frac{l'+m'}{2}, m')$;
  $complete(bottom)$;
  $P_{left} = P_{left} \cup (T_{leftgrounded} \times T_{bottom})$;
  $complete(left)$;
  $P_{right} = P_{right} \cup (T_{bottom} \times T_{rightgrounded})$;
  $complete(right)$;
  $P_{top} = P_{top} \cup (T_{leftgrounded} \times T_{right})$;
  $P_{top} = P_{top} \cup (T_{left} \times T_{rightgrounded})$;
  $complete(top)$

Листинг 1. Алгоритм Валианта
Listing 1. Valiant's algorithm

Сначала введем понятие перемножения двух подматриц матрицы разбора T.

Пусть $X \in (2^N)^{m \times l}, Y \in (2^N)^{l \times n}$ – две подматрицы T, тогда $X \times Y = Z$, где $Z \in (2^N)^{m \times n}$ и $Z_{i,j} = \bigcup_{k=1}^{l} X_{i,k} \times Y_{k,j}$.

Теперь можно представить $X \times Y = Z$ как перемножение $|N|^2$ булевых матриц (для каждой пары нетерминалов). Определим матрицу, соответствующую паре нетерминалов (B, C), как $Z^{(B,C)}$. $Z_{i,j}^{(B,C)} = 1$ тогда и только тогда, когда $(B, C) \in Z_{i,j}$. Также заметим, что $Z^{(B,C)} = X^B \times Y^C$. Более того, каждое из перемножений булевых матриц может быть обработано независимо.

С этими изменениями сложность алгоритма составляет $O(BMM(n)log(n))$, где $BMM(n)$ — время, необходимое для перемножения двух булевых матриц размера $n \times n$.

Алгоритм Валианта представлен на листинге 1. Все элементы матриц T и P инициализируются пустыми множествами. Затем эти элементы последовательно заполняются двумя рекурсивными процедурами.

Процедура $compute(l, m)$ корректно заполняет все $T_{i,j}$ для всех $l \leq i < j < m$.

Процедура $complete(l, m, l', m')$ заполняет все $T_{i,j}$ для всех $l \leq i < m$ и $l' \leq j < m'$. Для корректной работы этой процедуры, во-первых, необходимо, чтобы элементы $T_{i,j}$ для всех i и j, таких что $l \leq i < j \leq m$ и $l' \leq i < j \leq m'$ уже были построены. Во-вторых,

текущее значение $P_{i,j}$ для всех i и j, таких что $l \le i < m$ и $l' \le j < m'$, должно быть следующим:

$$P_{i,j} = \{(B,C): \exists k, (m \le k < l'), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)\}.$$

Процесс разбиения матрицы во время выполнения процедуры $complete(l,m,l',m')$ показан на рис. 1. Матрица в данном случае задана четверкой (l,m,l',m'), и она делится на четыре равные квадратные подматрицы: $bottom = (\frac{l+m}{2}, m, l', \frac{l'+m'}{2})$, $left = (l, \frac{l+m}{2}, l', \frac{l'+m'}{2})$, $right = (\frac{l+m}{2}, m, \frac{l'+m'}{2}, m')$, $top = (l, \frac{l+m}{2}, \frac{l'+m'}{2}, m')$. Кроме того, на рис.1 изображены две подматрицы, использующиеся в перемножении: $leftgrounded = \left(l, \frac{l+m}{2}, \frac{l+m}{2}, m\right)$, $rightgrounded = \left(l', \frac{l'+m'}{2}, \frac{l'+m'}{2}, m'\right)$.

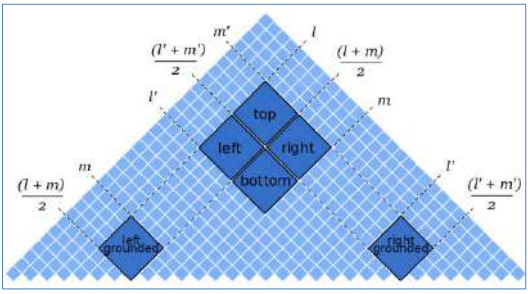

Рис. 1. Разбиение матриц, использованное в процедуре $complete(l,m,l',m')$
Fig. 1. Matrix partition used in $complete(l,m,l',m')$ procedure

На рис. 2 представлено начало процесса заполнения матрицы разбора. В следующем разделе будет приведен пример работы модификации, который наглядно продемонстрирует преимущества предложенной нами версии.

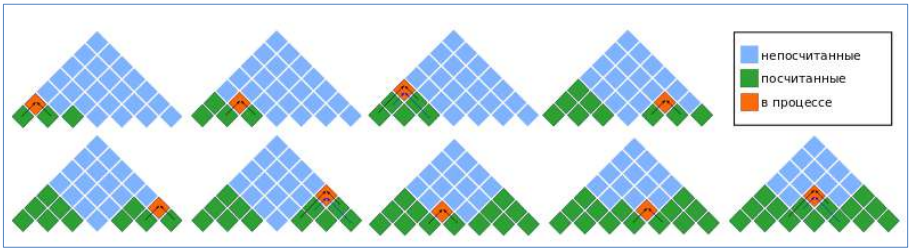

Рис. 2. Пример работы алгоритма Валианта
Fig. 2. An example of Valiant's algorithm

2.3 Задача поиска подстрок

При анализе генетических последовательностей в биоинформатике часто возникает задача нахождения одной или нескольких подпоследовательностей,

обладающих какими-либо характерными чертами. Наличие этих подпоследовательностей, а также их взаимное расположение позволяют получить важную информацию об исследуемом организме, например, о его происхождении. Часть подходов, используемых для их поиска, основаны на формальных грамматиках, с помощью которых можно эффективно описать эти черты. Главным недостатком таких подходов являются существенные проблемы с производительностью [16], которые можно решить с помощью алгоритма Валианта, но его трудно остановить на определенном этапе заполнения матрицы разбора и это потребует много лишних перемножений матриц.

3. Модификация алгоритма Валианта

В данном разделе мы представляем алгоритм, являющийся модификацией алгоритма Валианта. За счет изменения порядка вычисления подматриц предложенный алгоритм обладает такими практическими преимуществами, как возможность применения для решения задачи поиска подстрок и упрощение использования параллельных вычислений.

Главное отличие предложенного алгоритма – это возможность разделения матрицы разбора на слои непересекающихся подматриц одинакового размера. Пример разбиения матриц на такие слои представлен на рис. 3. Каждый слой состоит из квадратных подматриц, размер которых равен 2^n, где $n > 0$. Слои заполняются последовательно, снизу вверх, и каждая матрица слоя может обрабатываться независимо. На рис. 3 каждый слой выделен отдельным цветом, нижняя подматрица матрицы слоя ($bottom$), принадлежит предыдущему слою и уже заполнена. Таким образом, для каждой матрицы слоя нам необходимо заполнить только $left, right$ и top-подматрицы, поэтому такие слои мы будем называть V-образными.

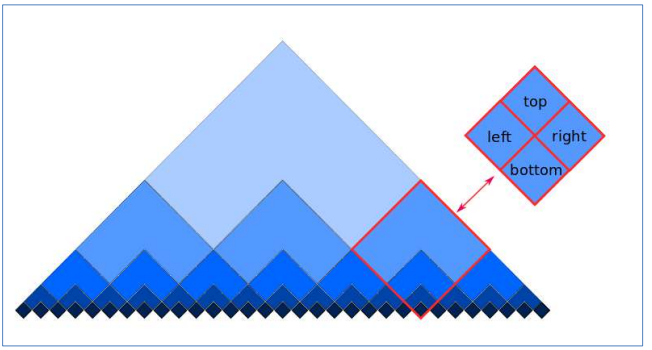

Рис. 3. Деление матриц на V-образные слои
Fig. 3. Matrix partition on V-shaped layers

Пример работы алгоритма показан на рис. 4. Нижний слой, состоящий из подматриц размера 1, вычисляется заранее, а заполнение матрицы начинается со второго слоя. (Здесь и далее, под слоем матриц будем понимать некоторое множество её подматриц разбора.) Также на рис. 4 на каждом шаге изображены операции, которые могут быть выполненные независимо, что позволяет значительно упростить разработку параллельной версии алгоритма.

Модификация алгоритма Валианта представлена на листинге 1. Процедура $main()$ заполняет нижний слой матрицы ($T_{l,l+1}$), а затем разделяет матрицу на слои так, как было описано ранее, и эти слои корректно вычисляются в процедуре $completeVLayer()$. Таким образом, вызов функции $main()$ заполнит всю матрицу разбора и вернет информацию о выводимости строки для заданной грамматики.

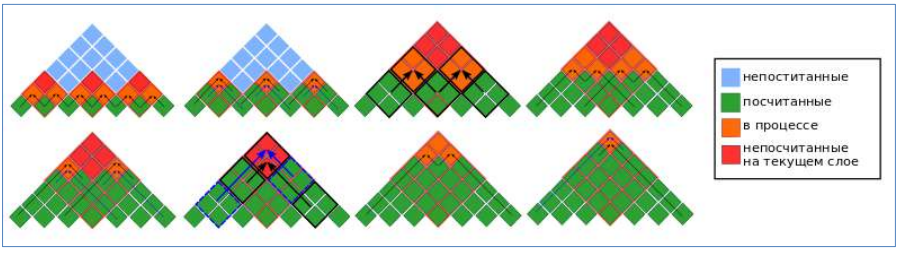

Рис. 4. Пример работы модификации алгоритма Валианта
Fig. 4. An example of the modification of Valiant's algorithm

Входные данные: G – КС-грамматика, $a = a_1 \dots a_n$, $a_i \in \Sigma, n \geq 1$, где $n + 1 = 2^p$
main():
 for $l \in \{1, \dots, n\}$ **do**
  $T_{l,l+1} = \{A\colon A \to a_i \in R\}$
 for $l \leq i < p - 1$ **do**
  $completeVLayer\ (constructLayer(i))$
 accept if and only if $S \in T_{0,n}$
$constructLayer\ (i)$:
 $\{(k2^i, (k+1)2^i, (k+1)2^i, (k+2)2^i) \mid 0 \leq k < 2^{p-i} - 1\}$
$completeLayer\ (M)$:
 if $\forall (l, m, l', m') \in M\ (m - l = 1)$ **then**
  for $(l, m, l', m') \in M$ **do**
   $T_{l,l'} = f(P_{l,l'})$
 else
  $completeLayer(bottomsublayer(M))$;
  $completeVLayer(M)$
complete$VLayer(M)$:
 $multiplicationTask_1 =$
  $\{left(subm), leftgrounded(subm), bottom(subm) \mid subm \in M\} \cup$
  $\{right(subm), bottom(subm), rightgrounded(subm) \mid subm \in M\}$;
 $multiplicationTask_2 =$
  $\{top(subm), leftgrounded(subm), right(subm) \mid subm \in M\}$;
 $multiplicationTask_3 =$
  $\{top(subm), left(subm), rightgrounded(subm) \mid subm \in M\}$;
 $performMultiplication(multiplicationTask_1)$;
 $completeLayer(leftsublayer(M) \cup rightsublayer(M))$;
 $performMultiplication(multiplicationTask_2)$;
 $performMultiplication(multiplicationTask_3)$;
 $completeLayer(topsublayer(M)$
performMultiplication($tasks$):
 for $(m, m1, m2) \in tasks$ do
  $P_m = P_m \cup (T_{m1} \times T_{m2})$

Листинг 2. Модификация алгоритма Валианта
Listing 2. Modification of Valiant's algorithm

Для краткости, аналогично тому, как сделано в алгоритме Валианта, определим дополнительные функции: $left(subm)$, $right(subm)$, $bottom(subm)$, $top(subm)$, $rightgrounded(subm)$ and $leftgrounded(subm)$, которые возвращают различные подматрицы для матрицы $subm = (l, m, l', m')$. Также определим функции для слоя подматриц M:

- $bottomsublayer(M) = \{bottom(subm) \mid subm \in M\}$,
- $leftsublayer(M) = \{left(subm) \mid subm \in M\}$,
- $rightsublayer(M) = \{right(subm) \mid subm \in M\}$,
- $topsublayer(M) = \{top(subm) \mid subm \in M\}$.

Процедура $completeVLayer(M)$ принимает на вход слой непересекающихся подматриц одинакового размера M. Для каждой $subm = (l, m, l', m') \in M$ эта процедура вычисляет $left(subm)$, $right(subm)$, $top(subm)$. Для корректной работы этой процедуры, во-первых, необходимо, чтобы элементы $bottom(subm)$ и $T_{i,j}$ для всех i и j, таких что $l \leq i < j \leq m$ и $l' \leq i < j \leq m'$, уже были построены. Во-вторых, текущее значение $P_{i,j}$ для всех i и j, таких что $l \leq i < m$ и $l' \leq j < m'$, должно быть следующим:

$$P_{i,j} = \{(B, C)\colon \exists k, (m \leq k < l'), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)\}.$$

Процедура $completeLayer(M)$ также принимает на вход набор подматриц M, но каждую $subm \in M$ заполняет полностью. Ограничения на входные данные у этой процедуры такие же, как и у процедуры $completeVLayer()$, за исключением ограничения на $bottom(subm)$, которое в данном случае не требуется.

Другими словами, $completeVLayer(M)$ вычисляет весь V-образный слой M, а $completeLayer(M_2)$ – вспомогательная функция, необходимая для вычисления меньших квадратных подматриц слоя M.

Наконец процедура $performMultiplication(tasks)$, где $tasks$ – это массив троек подматриц, реализует основной шаг алгоритма – перемножение матриц. Стоит заметить, что в отличие от исходного алгоритма Валианта, в данном случае $|tasks| \geq 1$ и каждый $task \in tasks$ может быть выполнен параллельно.

Для представленного алгоритма докажем следующие утверждения.

Лемма 1. Пусть M – слой и для всех $(l, m, l', m') \in M$ справедливо следующее:

- для всех i и j, таких что $l \leq i < j \leq m$ и $l' \leq i < j \leq m'$,
$$T_{i,j} = \{A: \ a_{i+1} \dots a_j \in L_G(A)\};$$
- для всех i и j, таких что $l \leq i < m$ и $l' \leq j < m'$,
$$P_{i,j} = \{(B, C): \exists k, (m \leq k < l'), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)\}.$$

Тогда процедура $completeLayer(M)$ возвращает корректно заполненные $T_{i,j}$ для всех i и j, таких что $l \leq i < m$ и $l' \leq j < m'$ при этом $(l, m, l', m') \in M$.

Доказательство.

Индукция по размеру матриц в слое M. ■

Теорема 1. (*Корректность алгоритма*). Алгоритм, представленный на листинге 2, корректно заполняет $T_{i,j}$ для всех i и j, и входная строка $a_1 \dots a_n \in L_G(S)$ тогда и только тогда, когда $S \in T_{0,n}$.

Доказательство.

Перед тем, как доказать утверждение теоремы, докажем по индукции, что все слои матрицы разбора T вычисляются корректно.

База индукции. Слой размера 1×1 корректно заполняется в строках 2-3 листинга 2.

Индукционный переход. Предположим, что все слои размера $\leq 2^{p-2} \times 2^{p-2}$ вычислены корректно.

Обозначим слой размера $2^{p-1} \times 2^{p-1}$ как M. Будем рассматривать одну матрицу слоя $subm = (l, m, l', m')$, так как заполнение остальных подматриц производится аналогично.

Рассмотрим вызов процедуры $completeVLayer(M)$. Заметим, что все $T_{i,j}$ для всех i и j, таких что $l \leq i < j < m$ и $l' \leq i < j < m'$, уже корректно заполнены, так как эти элементы лежат в слоях, которые уже вычислены по индукционному предположению.

В начале выполнения процедуры $completeVLayer(M)$, $performMultiplication(multiplicationTask_1)$ добавляет к каждому $P_{i,j}$ все пары нетерминалов (B, C), такие что $\exists k, \left(\frac{l+m}{2} \leq k < l'\right), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)$ для всех $(i, j) \in leftsublayer(M)$ и все пары (B, C), такие что $\exists k, \left(m \leq k < \frac{l'+m'}{2}\right), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)$ для всех $(i, j) \in rightsublayer(M)$. Теперь все предусловия для вызова процедуры $completeLayer(leftsublayer(M) \cup rightsublayer(M))$ выполнены и она вернет корректно заполненные $leftsublayer(M) \cup rightsublayer(M)$.

Затем функция $performMultiplication$ вызывается от аргументов $multiplicationTask_2$ и $multiplicationTask_3$ и к каждому $P_{i,j}$ добавляет пары нетерминалов (B, C), такие что $\exists k, \left(\frac{l+m}{2} \leq k < m\right), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)$ и (B, C), такие что $\exists k, \left(l' \leq k <\right.$

$\frac{l'+m'}{2}\Big), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)$ для всех $(i,j) \in topsublayer(M)$. Так как $m' = l$ (из построения слоя), условия на матрицу P выполнены и $completeLayer(topsublayer)$ вернет корректно заполненный $topsublayer(M)$.

Таким образом, $completeVLayer(M)$ для каждого слоя M возвращает корректные $T_{i,j}$ для всех $(i,j) \in M$ матрицы T и строки 4-6 листинга 2 возвращают все $T_{i,j} = \{A: A \in N, a_{i+1} \dots a_j \in L_G(A)\}$. ■

Лемма 2. Пусть calls_i – это количество вызовов процедуры $completeVLayer(M)$, где для всех $(l,m,l',m') \in M$ выполнено $m - l = 2^{p-i}$. Тогда истинны следующие утверждения:

- для всех $i \in \{1,\dots,p-1\}$ $\sum_{n=1}^{calls_i} |M| = 2^{2i-1} - 2^{i-1}$;
- для всех $i \in \{1,\dots,p-1\}$ матрицы размера $2^{p-i} \times 2^{p-i}$ перемножаются ровно $2^{2i-1} - 2^i$ раз.

Доказательство.

Сначала докажем первое утверждение индукцией по i.

База индукции. При $i = 1$: $calls_1 = 1$ и $|M| = 1$. Следовательно, $2^{2i-1} - 2^{i-1} = 2^1 - 2^0 = 1$.

Индукционный переход. Предположим, что $\sum_{n=1}^{calls_i} |M| = 2^{2i-1} - 2^{i-1}$ для всех $i \in \{1,\dots,j\}$. Пусть $i = j + 1$.

Заметим, что функция $constructLayer(i)$ возвращает $2^{p-i} - 1$ матриц размера $2^i \times 2^i$, то есть при вызове процедуры $completeVLayer(constructLayer(k-i))$ процедура $constructLayer(k-i)$ вернет $2^i - 1$ матриц размера $2^{p-i} \times 2^{p-i}$. Также, $completeVLayer(M)$ будет вызван 3 раза для левых, правых и верхних подматриц матриц размера $2^{p-(i-1)} \times 2^{p-(i-1)}$. Кроме того, $completeVLayer(M)$ вызывается 4 раза для нижних, левых, правых и верхних подматриц матриц размера $2^{p-(i-2)} \times 2^{p-(i-2)}$, за исключением левых, правых и верхних подматриц размера $2^{i-2} - 1$ матриц, которые к этому моменту уже были посчитаны.

Таким образом, $\sum_{n=1}^{calls_i} |M| = 2^i - 1 + 3 * (2^{2(i-1)-1} - 2^{(i-1)-1}) + 4 * (2^{2(i-2)-1} - 2^{(i-2)-1}) - (2^{i-2} - 1) = 2^{2i-1} - 2^{i-1}$.

Теперь мы знаем, что $\sum_{n=1}^{calls_{i-1}} |M| = 2^{2(i-1)-1} - 2^{(i-1)-1}$, и можем доказать второе утверждение. Посчитаем количество перемножений матриц размера $2^{p-i} \times 2^{p-i}$. Функция $performMultiplication$ вызывается 3 раза, $|multiplicationTask_1| = 2 * (2^{2(i-1)-1} - 2^{(i-1)-1})$ и $|multiplicationTask_2| = |multiplicationTask_3| = 2^{2(i-1)-1} - 2^{(i-1)-1}$. То есть, количество перемножений подматриц размера $2^{p-i} \times 2^{p-i}$ равно $4 * (2^{2(i-1)-1} - 2^{(i-1)-1}) = 2^{2i-1} - 2^i$. ■

Теорема 2. (*Оценка сложности алгоритма*). Пусть $|G|$ – длина описания грамматики G и n – длина входной строки. Тогда алгоритм, представленный на листинге 2, заполняет матрицу T за $O(|G|BMM(n)log(n))$, где $BMM(n)$ — время, необходимое для перемножения двух булевых матриц размера $n \times n$.

Доказательство.

Так как в лемме 2 было показано, что количество перемножений матриц не изменилось по сравнению с исходной версией алгоритма Валианта, то доказательство будет идентично доказательству теоремы 1 в [8]. ■

Таким образом, мы доказали корректность предложенного алгоритма, а также показали, что его сложность осталась такой же, как сложность исходного алгоритма Валианта.

4. Применение алгоритма к задаче поиска подстрок

В данном разделе мы продемонстрируем, как предложенный алгоритм может быть применена к задаче поиска подстрок.

Пусть для входной строки размера $n = 2^p$ мы хотим найти все подстроки размера s, которые принадлежат языку, заданному грамматикой G. Тогда мы должны посчитать слои подматриц, размер которых не превышает 2^r, где $2^{r-2} < s \leq 2^{r-1}$.

Пусть $r = p - (m-2)$ и, следовательно, $m - 2 = p - r$.

Для всех $m \leq i \leq p$ перемножение матриц размера $2^{p-i} \times 2^{p-i}$ выполняется ровно $2^{2i-1} - 2^i$ раз и каждое из них включает перемножение $C = O(|G|)$ булевых подматриц.

$$C * \sum_{i=m}^{p} 2^{2i-1} * 2^{\omega(p-i)} * f(2^{p-i}) = C * 2^{\omega r} * \sum_{i=2}^{r} 2^{(2-\omega)i} * 2^{2(p-r)-1} * f(2^{r-i})$$

$$\leq C * 2^{\omega r} * f(2^r) * 2^{2(p-r)-1} * \sum_{i=2}^{r} 2^{(2-\omega)i} = BMM(2^r) * 2^{2(p-r)-1} * \sum_{i=2}^{r} 2^{(2-\omega)i}$$

Временная сложность алгоритма для поиска всех подстрок длины s равна $O(2^{2(p-r)-1}|G|BMM(2^r)log(n))$, где дополнительный множитель обозначает количество матриц в последнем вычисленном слое. Однако этот множитель, во-первых, мал относительно общей по времени работы алгоритма, во-вторых, не существенен, так как эти матрицы могут быть обработаны параллельно. Алгоритм Валианта, в отличие от модификации, не может так легко быть применен к данной задаче. В нем будет необходимо полностью вычислить, как минимум, две треугольные подматрицы размера $\frac{n}{2}$ (как показано на рис. 5). Это значит, что в любом случае не получится добиться сложности лучше, чем $O(|G|BMM(2^{p-1})log(p-2))$.

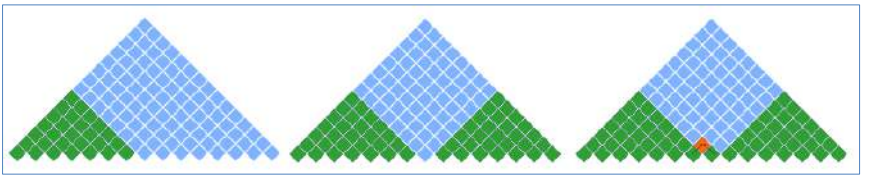

Рис. 5. Количество элементов, вычисляемых в алгоритме Валианта (2 треугольные подматрицы размера $\frac{n}{2}$, выделенные зеленым цветом)

Fig. 5. Elements computed in Valiant's algorithm (at least 2 triangle submatrices of size $\frac{n}{2}$, which are marked in green).

В завершение данного раздела скажем, что предложенный алгоритм может быть эффективно применен для строк размера $s \ll n$, что и было показано в проведённых экспериментах.

5. Эксперименты

В этом разделе мы приводим результаты экспериментов, целью которых является демонстрация практической применимости предложенного алгоритма к задаче поиска подстрок.

5.1 Постановка экспериментов

Эксперименты проводились на рабочей станции со следующими характеристиками: операционная система — Linux Mint 19.1, ЦПУ — Intel i5-8250U, 1600-3400 Mhz, 4 Core(s), 8 Logical Processor(s), оперативная память — 8 GB.

Была выполнена реализация исходного алгоритма Валианта и предложенного алгоритма на языке программирования C++. Для перемножения подматриц использовалась библиотека для высокоэффективной работы с булевыми матрицами M4RI [9].

Основной целью поставленных экспериментов было исследование эффективности предложенного алгоритма.

Для этого были поставлены следующие вопросы:

RQ1. *Сравнение предложенного алгоритма с исходным алгоритмом Валианта.*

RQ2. *Эффективность применения предложенного алгоритма для задачи поиска подстрок.*

Для сравнительного анализа алгоритмы были реализованы в соответствии с листингами 1 и 2. При исследовании эффективности предложенного алгоритма к задаче поиска подстрок была изменена функция $main()$ из листинга 2: теперь она принимает дополнительный аргумент s – длину максимальной искомой подстроки и вычисляет только те слои, которые содержат в себе нужные подстроки, как было показано в предыдущем разделе.

Для экспериментов использовалась КС-грамматика D_2, порождающая язык Дика с двумя видами скобок и стартовым нетерминалом S [17]. Правила данной грамматики представлены на рис. 6. [Привлеките внимание читателя с помощью яркой цитаты из документа или используйте это место, чтобы выделить ключевой момент. Чтобы поместить это текстовое поле в любой части страницы, просто перетащите его.]

$$S \rightarrow SS \mid (S) \mid [S] \mid \varepsilon$$

Рис. 6. Грамматика D_2
Fig. 6. Grammar D_2

Эта грамматика была выбрана потому, что грамматики для описания правильных скобочных последовательностей часто применяются при анализе строк в биоинформатике.

Грамматика D_2 переводится в нормальную форму Хомского и подается на вход алгоритму со специально сгенерированными строками различной длины (127-8191 символов). Строки составлены следующим образом: заранее создается подстрока, принадлежащая языку Дика, далее в полную строку вставляется максимально возможное количество созданных подстрок, которые можно разделить “перегородками” (терминалами, из-за которых все остальные строки, кроме вставленных, будут невыводимыми в грамматике D_2). Строки были созданы таким образом, чтобы проверять корректность предложенного алгоритма.

5.2 Анализ результатов

RQ1. *Сравнение предложенного алгоритма с исходным алгоритмом Валианта.*

Результаты сравнительного анализа алгоритма Валианта и его модификации представлены в табл. 1, где N – длина сгенерированной строки. Для двух реализаций – алгоритма Валианта (VAL) и предложенного алгоритма (MOD) представлено время работы в миллисекундах.

Результаты сравнительного анализа (см. табл. 1) показывают, что предложенная модификация и исходный алгоритм Валианта работают практически одинаково.

RQ2. *Эффективность применения предложенного алгоритма для задачи поиска подстрок.*

Результаты работы адаптированной к задаче поиска подстрок модификации представлены в табл. 2, где N – длина сгенерированной строки, s – длина искомых подстрок. Для алгоритма Валианта (VAL) и модификации с измененной функцией $main()$ (ADP) представлено время работы в миллисекундах.

Табл. 1. Результаты сравнительного анализа
Table 1. Evaluation results for comparative analysis

N	VAL	MOD
127	78	76
255	289	292
511	1212	1177
1023	4858	4779
2047	19613	19279
4095	78361	78279
8191	315677	315088

Табл. 2. Результаты работы алгоритма для задачи поиска подстрок
Table 2. Evaluation results for string-matching problem

S	N	VAL	ADP
250	**1023**	4858	2996
250	**2047**	19613	6649
510	**2047**	19613	12178
250	**4095**	78361	13825
510	**4095**	78361	26576
1020	**4095**	78361	48314
250	**8191**	315677	28904
510	**8191**	315677	56703
1020	**8191**	315677	108382
2040	**8191**	315677	197324

Результаты второго эксперимента (см. табл. 2) показывают, что адаптированная версия модификации может быть эффективно применена к задаче поиска подстрок: она корректно находит все выводимые подстроки в строке и работает существенно быстрее алгоритма Валианта, который совершает большое количество лишних вычислений из-за сложности его преждевременной остановки.

Таким образом, поставленные эксперименты демонстрируют практическую применимость предложенного алгоритма.

6. Заключение

В данной работе был предложен алгоритм, являющейся модификацией алгоритма Валианта, который обладает некоторыми преимуществами по сравнению с исходной версией. За счет разбиения исходной матрицы на слои подматриц появилась возможность останавливать работу алгоритма, не вычисляя всю матрицу разбора до конца, если этого требует поставленная задача. Проведенные эксперименты показали, что предложенный алгоритм не проигрывает в производительности исходной версии алгоритма. Более того, была показана применимость данной модификации к задаче поиска подстрок.

Определим несколько направлений для будущих исследований. Так как все подматрицы в слое могут быть обработаны независимо, необходимо проверить, насколько эффективно могут быть применены техники параллельных вычислений. Кроме того, открытым остается вопрос, можно ли как-нибудь еще изменить порядок перемножения подматриц, чтобы полностью избавить алгоритм от рекурсивных вызовов?

Список литературы / References

[1]. Okhotin A. Conjunctive grammars. Journal of Automata, Languages and Combinatorics, vol. 6, no. 4, 2001, pp. 519-535.
[2]. Chomsky N. On certain formal properties of grammars. Information and control, vol. 2, no. 2, 1959, pp. 137-167.
[3]. Kasami T. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, 1965.
[4]. Younger D.H. Recognition and parsing of context-free languages in time n3. Information and control, vol. 10, no. 2, 1967, pp. 189-208.
[5]. Valiant L.G. General context-free recognition in less than cubic time. Journal of computer and system sciences, vol. 10, no. 2, 1975, pp. 308-315.
[6]. Okhotin A. Conjunctive and Boolean grammars: the true general case of the context-free grammars. Computer Science Review, vol. 9, 2013, pp. 27-59.
[7]. Okhotin A. Boolean grammars. Information and Computation, vol. 194, issue 1, 2004, pp. 19-48.
[8]. Okhotin A. Parsing by matrix multiplication generalized to Boolean grammars. Theoretical Computer Science, vol. 516, 2014, pp. 101-120.
[9]. Albrecht M., Bard G. The M4RI Library. The M4RI Team. 2019. Available at: https://bitbucket.org/malb/m4ri, accessed 20.01.2010.
[10]. Earley J. An effcient context-free parsing algorithm. Communications of the ACM, vol. 13, no. 2, 1970, pp.94-102
[11]. Bernardy J.P., Claessen K. Effcient divide-and-conquer parsing of practical context-free languages. ACM SIGPLAN Notice, vol. 48, no. 9, 2013, pp.111-122
[12]. Grigorev S., Lunina P. The Composition of Dense Neural Networks and Formal Grammars for Secondary Structure Analysis. In Proc. of the 12th International Joint Conference on Biomedical Engineering Systems and Technologies, vol. 3, 2019, pp. 234-241.
[13]. Rivas E., Eddy S.R. The language of RNA: a formal grammar that includes pseudoknots. Bioinformatics, vol. 16, no. 4, 2000, pp. 334-340.
[14]. Knudsen B. and Hein J. Rna secondary structure prediction using stochastic context-free grammars and evolutionary history. Bioinformatics, vol. 15, no. 6, 1999, pp. 446-454.
[15]. Dowell R.D., Eddy S.R. Evaluation of several lightweight stochastic context-free grammars for rna secondary structure prediction. BMC bioinformatics, vol. 5, no. 1, 2004, Article number: 71.
[16]. Durbin R., Eddy S., Krogh A., and Mitchison G. Biological sequence analysis. Cambridge University Press, 1996, 356 p.
[17]. Hopcroft J.E., Ullman J.D. Formal languages and their relation to automata. Addison-Wesley, 1969, 242 p.

Информация об авторах / Information about authors

Юлия Алексеевна СУСАНИНА получила степень бакалавра в Санкт-Петербургском государственном университете по направлению Математическое обеспечение и администрирование информационных систем. Ее области научных интересов включают теорию формальных языков и ее применения.

Yuliya Alekseevna SUSANINA received a Bachelor degree in Saint Petersburg State University on Software and Administration of Information Systems programme. Her research interests include formal language theory and its applications.

Анна Никитична ЯВЕЙН – получила степень бакалавра в Санкт-Петербургском государственном университете по направлению Прикладная математика и информатика. Ее области научных интересов включают статический анализ кода и компиляторы.

Anna Nikitichna YAVEYN received a Bachelor degree in Saint Petersburg State University on Applied Mathematics and Computer Science Systems programme. Her research interests include static code analysis and compilers.

Семен Вячеславович ГРИГОРЬЕВ – кандидат физико-математических наук, работает в Санкт-Петербургском государственном университете. В 2016 году защитил кандидатскую диссертацию на тему "Синтаксический анализ динамически формируемых программ". Области научных интересов: теория формальных языков и её приложения, графовые базы данных, статический анализ кода, параллельные вычисления.

Semyon Vyacheslavovich GRIGOREV is PhD in Physical and Mathematical Sciences in Saint Petersburg State University. He defended his PhD thesis «Parsing of dynamically generated code» in 2016. His research interests include formal language theory and applications, graph databases, static code analysis and parallel computing.

DOI: 10.15514/ISPRAS-2020-32(2)-12

HP-Graph as a Basis of a DSM Platform Visual Model Editor

N.M. Suvorov, ORCID: 0000-0003-2871-9757 <SuvorovNM@gmail.com>
L.N. Lyadova, ORCID: 0000-0001-5643-747X <LNLyadova@gmail.com>
National Research University Higher School of Economics,
38 Studencheskaya Ulitsa, Perm, 614070, Russia

Abstract. The language-oriented approach is becoming more and more popular in the development of information systems, but the existing DSM platforms that implement this paradigm have significant limitations, including insufficient expressive capabilities of the models used to implement visual model editors for complex subject areas and limited abilities to transform visual models. Visual languages are usually based on graph models, but the types of graphs used have certain limitations, such as insufficient expressiveness, the complexity of representing large-dimensional models and operation executions. For creating a tool that does not have the described constraints, development of a new formal model is needed. *HP*-graphs can become a solution for this problem. It is not only possible to create new visual languages for diverse domains based on them, but also to develop efficient algorithms to perform different operations on models constructed using these languages. The *HP*-graph definition is given and the justification of the expressive power of the proposed model is presented, the main operations for HP-graphs are described. The chosen graph formalism combines the capabilities of different types of graphs to represent visual models and allows creating a flexible model editor for the DSM platform, to implement effective algorithms of performing operations, in particular, model transformations.

Keywords: Domain-specific language; DSM platform; visual model; graph model; HP-graph; algorithms on graphs.

For citation: Suvorov N.M., Lyadova L.N. HP-Graph as a Basis of a DSM Platform Visual Model Editor. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 149-160. DOI: 10.15514/ISPRAS–2020–32(2)–12

HP-граф как основа для разработки редактора визуальных моделей DSM-платформы

Н.М. Суворов, ORCID: 0000-0003-2871-9757 <SuvorovNM@gmail.com>
Л.Н. Лядова, ORCID: 0000-0001-5643-747X <LNLyadova@gmail.com>
Национальный исследовательский университет «Высшая школа экономики»,
614070, Россия, г. Пермь, ул. Студенческая, д. 38.

Аннотация. Языково-ориентированный подход становится все более популярным при разработке информационных систем, однако существующие *DSM*-платформы, реализующие эту парадигму, имеют существенные ограничения, включающие недостаточные выразительные возможности моделей, используемых для реализации редакторов визуальных моделей для сложных предметных областей, и ограниченные возможности для трансформации визуальных моделей. Визуальные языки обычно основаны на графовых моделях, однако используемые типы графов имеют определенные ограничения, такие как недостаточная выразительность, сложность представления моделей большой размерности, а также трудоёмкость выполнения операций. Для создания инструмента, не имеющего описанных выше ограничений, необходима разработка новой формальной модели. *HP*-графы могут стать решением данной проблемы. Имеется не только возможность создавать новые визуальные языки для различных предметных областей на их основе, но и разработать эффективные алгоритмы для выполнения

различных операций над моделями, построенными с использованием этих языков. В статье дано определение *HP*-графа, а также приведено обоснование выразительной мощности предложенной модели, описаны основные операции для *HP*-графов. Выбранный графовый формализм объединяет возможности различных типов графов для представления визуальных моделей и позволяет создать на его основе гибкий редактор моделей для *DSM*-платформы, реализовать эффективные алгоритмы выполнения операций, в частности, трансформации моделей.

Ключевые слова: предметно-ориентированный язык; DSM платформа; визуальная модель; графовая модель; HP-граф; алгоритмы на графах.

Для цитирования: Суворов Н.М., Лядова Л.Н. HP-граф как основа для разработки редактора визуальных моделей DSM-платформы. Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 149-160 (на английском языке). DOI: 10.15514/ISPRAS–2020–32(2)–12

1. Introduction

The study of any objects and processes, as well as their design, can barely be done without modeling, that is why software tools that allow non-IT specialists to build various models and formalize descriptions of objects and processes, or use modeling as a method of analysis for the study of objects are becoming more popular. Among the subject areas where modeling is particularly important, the development of information systems stands out. Currently the main approach to creating large information systems is a model-oriented approach [1]. Using it, developers usually deal only with models, which helps to ensure high quality of programs and prevent errors. CASE tools [2], which automate the system development process as much as possible due to the capabilities of visual modeling, model interpretation, and code generation based on the created models, are used especially for these purposes.

However, the traditional model-oriented approach to developing systems has its drawbacks, among which are:

- universality of the languages used for system development as languages operate not in terms of the subject area, but in constructs of the means by which the system is created;
- immutability of modeling languages, which does not allow all the subtleties, pitfalls, and limitations of the subject area to be displayed and taken into account;
- complexity of modification of the created systems as making changes to the system is possible only if there are development tools, source codes and a professional IT specialist;
- the impossibility of transitioning from one modeling language to another, but, creating large systems usually involves building several models describing the system from different points of view, with different granularity, so in such cases, there is a need to harmonize the models created by different professionals at different stages of development, which requires the ability to perform a transition from one modeling language to another.

These problems are solved by a paradigm called language-oriented programming [3]. This paradigm at the initial stage of development implies the creation of a metamodel of a subject area represented by one or more languages for solving various project tasks. These languages are used to build the necessary models for implementing the system. For implementing this approach DSM-platforms [4], language tools, and Meta-CASE systems, such as MetaEdit+ [5], which facilitate the development of domain-specific languages (DSL), are usually used. These languages operate in terms of the subject area and reflect the specifics of the tasks they solve. Moreover, subject-oriented languages can also consider the qualifications of users who will use them [6].

Nevertheless, existing tools only partially solve the problems of the traditional approach to modeling. To solve all the problems described above, a language tool must meet the particular requirements [4], [7], [8]. It should

- have an ability to define modeling languages for most subject areas;
- have an ability to dynamically change the modeling language;

- have an ability to alienate the created modeling language from the system where it has been developed;
- have an ability to modify the visual model of the system, rather than the source code, when a modeled process or system undergoes changes;
- unify representation and description of both models and metamodels, which allows a person to work with models and metamodels using the same tools, as well as, for example, provides the opportunity to perform vertical and horizontal transformations of visual models.

To create a tool that has all these features, the development of a new formal model is needed. Visual languages are usually based on graph models [1], but the types of graphs used have limitations, such as insufficient expressiveness of the created models, inefficiency, and complexity of operations. However, there is a more powerful formal model that solves these problems, but has not been used by developers yet, which is called a hypergraph with poles (*HP*-graph) [9], which connects the expressive capabilities of various types of graph models.

2. Related works

Many different tools have been created that allow people to develop modeling languages and build models based on these languages. These tools are Microsoft DSL Tools [10], Eclipse Sirius [11], MetaEdit+ [5], Microsoft Visio [12], QReal [13], etc. Detailed description and comparison of these platforms is given in [4]. All these platforms have some limitations and do not fully meet the requirements described above. Consideration of main constraints of the platforms is given below.

Microsoft DSL Tools uses templates based on UML diagrams to create a new DSL, which leads to complexity and confusion when building model hierarchies and leads to appearance of limitations and inaccuracies in the resulting modeling language [8]. Moreover, this platform is characterized by the lack of the ability to dynamically change metamodels and transform models, as well as the inability to use DSLs outside of MS Visual Studio.

Eclipse Sirius offers a solution for rapid development of a graphical tool for DSM, but certain complex tasks may require changes to the EMF and GMF code. There is also a need for interpreted expressions which will be evaluated at runtime to provide a behavior specific to domain and representations and which can only be written in Acceleo, OCL or Java [11]. Sirius allows a user to perform horizontal transformations, but the knowledge of special addons is needed.

MetaEdit+ contains only limited possibilities for transforming visual models. Models exported from the platform have their own format, which makes it difficult to use models created in this platform in other software tools.

The main drawbacks of Microsoft Visio are the inability to change the modeling language while the system is running, and the need to purchase MS Visio to use the tools developed on its basis. Also, a language metamodel can only be built using a UML class diagram, which significantly limits the platform's capabilities and complicates the process of creating languages.

The QReal platform does not have the ability to change the metalanguage, the ability to transform models, and this platform is characterized by the complexity of modifying the created modeling language.

As it seems from the Table 1, there is no platform that meets all the previously given requirements. Nevertheless, it should be noted that at least some of the requirements for tools are met by each of the platforms listed.

Table 1. The comparison of the tools

Requirements	MS DSL Tools	Eclipse Sirius	Meta Edit+	MS Visio	QReal
Ability to define modeling languages for most subject areas	+	+	+	+	+
Ability to dynamically change the modeling language	–	–	+	–	–
Ability to alienate the created language from the system	–	+	–	–	–
Ability to modify the visual model	+	+	+	+	+
Ability to perform a horizontal transformation	–	+	–	–	–

Modeling and designing information systems tend to be done using special methodological approaches which can be divided to structural and object-oriented approaches. Despite the difference in the approaches and the division of all tools into two large groups depending on the approach underlying them (UML and "No-UML"), there are general modeling principles that the model should be aimed to implement.

The essence of the structural approach is to decompose a process into automated functions – the function of the upper level is decomposed and divided into subfunctions, refining properties of the functions at the upper levels of the hierarchy. Each subfunction, in turn, is decomposed into elements of the next level, and this happens until the obtained structure becomes trivial enough. Among the diagrams of this approach are Structural Analysis and Design Technique (SADT), a Data-Flow Diagram (DFD) and an Entity-Relation Diagram (ERD). The structural approach is used in simulation systems [15], as well as for functional and information modeling [16]. DSL can also be developed as part of this approach [17].

The essence of the object-oriented approach is an object decomposition, when the system is represented as a set of objects that exchange messages during the interaction. Moreover, the object itself in this case is an independent entity characterized by its state, behavior, and semantics [18]. Based on this approach, a set of DSLs [19],[20] is developed, but this approach is characterized by certain disadvantages, among which the complexity of building a hierarchy of models is highlighted. Using this approach does not always allow a person to properly express the concepts of the subject area, so the resulting language may have some limitations and inaccuracies. However, using it we can significantly reduce the language development time [21]. With all this in mind, the formalism underlying the visual model editor for a DSM-platform must meet the following requirements:

- to allow multi-level and multi-aspect modeling, which makes the decomposition of models from different points of view possible;
- to unify the description of models at different levels of the hierarchy, which means that the same formalism should be used to describe both models and metamodels;
- to allow development of modeling languages for a wide range of subject areas;
- to allow a user to discard constructions that are not details of the subject area, which will simplify the study of the developed language by end users;
- to perform both horizontal and vertical transformations.

Various types of graph formalisms are used for constructing and visualizing models, including oriented graphs, multigraphs [22], hypergraphs [23], hi-graphs [24], [25], meta-graphs [7], [26], *P*-graphs [27], [28]. Nevertheless, all these formalisms cannot meet all the mentioned requirements due to their certain limitations, therefore, development of a new graph model is needed.

3. Description of the graph model

Hypergraph with poles (*HP*-graph) is a graph model which meets the given requirements and can be used as a base for a visual model editor.

HP-graph is an ordered triple $G = (P, V, W)$, where $P = \{\pi_1,\ldots,\pi_n\}$ is a set of external poles, $V = \{v_1,\ldots,v_m\}$ is a non-empty set of vertices, $W = \{w_1,\ldots,w_l\}$ is a set of edges [9]. Let *Pol* be an abstract set of all poles of the graph. Thus, 2^{Pol} is a powerset of all poles of the graph. Then:

- Every vertex $v \in V$ is a subset of the set of all subsets of poles ($v \subset 2^{Pol}$) but $\forall v_i \in V, \forall v_j \in V$ $[i \neq j \rightarrow v_i \cap v_j = \varnothing]$, which means that V is a set of mutually disjoint subsets of *Pol*.
- A set of external poles P is also a subset of the powerset of poles ($P \subset 2^{Pol}$). This set consists of input and output poles of the graph ($P = I(G) \cup O(G)$). Each vertex of the graph $v \in V$ is also represented by a set of input ($I(v)$) and output ($O(v)$) poles $P_v = \{p_{v_1},\ldots p_{v_n}\}$ ($\forall v \in V\ \exists I(v) \subset P_v, \exists O(v) \subset P_v [I(v) \cup O(v) = v]$). Sets of input and output poles can also intersect. If no poles are specified for a vertex, it is assumed that the vertex consists of a single pole, which is both input and output ($I(v) = O(v)$).
- Each edge $w \in W$ defines connections between vertices and is represented as a subset of the powerset of poles ($w = P_w = \{p_{w_1},\ldots,p_{w_k}\} \subset 2^{Pol}$). An edge cannot be represented as an empty set ($\forall w \in W$: $[P_w \neq \varnothing]$). The edge can allow a vertex to be even linked to itself. Each edge must contain at least one input pole and one output pole, so for $\forall v \in V(G)$, $\forall w \in W(G)$ the following condition must be met: $[\exists p \in w \cap (I(v) \cup I(G))$ *and* $\exists r \in w \cap (I(v) \cup I(G))]$.

An example of the hypergraph with poles is demonstrated in fig. 1.

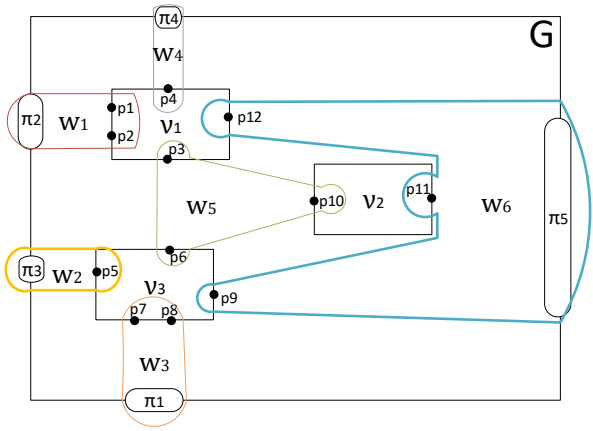

Fig. 1. Example of an HP-graph

In this figure, external poles are represented by a set $P = \{\pi_1,\ldots,\pi_5\}$, edges are represented by a set $W = \{w_1,\ldots,w_6\}$ and vertices are represented by a set $V = \{v_1, v_2, v_3\}$. Every vertex contains a certain number of poles *p*, which are connected to other poles by means of hyperedges from the set *W*.

In the *HP*-graph, edges and vertices are represented as sets of inputs and outputs, while the actual structure of these elements is hidden. Thus, it can be assumed that these elements are represented as a «Black box».

3.1 Main operations

To describe main operations on the *HP*-graph, let us define $G = (P, V, W)$ as an original *HP*-graph, $G' = (P', V', W')$ as a resulting *HP*-graph, *v* as a vertex, *p*1 as an inner pole, *p*2 as an outer pole and *w* as an edge.

The following operations add elements to an HP-graph:

- $v + p1$ is the addition of the inner pole to the node. The pole is added to both the vertex itself and the set of all poles of the graph:

$$Pol(G') = Pol(G) \cup \{p1\},$$
$$v' = v \cup \{p1\}.$$

- $G + v$ is the addition of the node to the graph. If a cardinality of v is more than 0 ($|v|>0$), a vertex is added to the set of vertices $V(G)$, and all poles of this vertex are added to the set of all poles of the graph:

$$Pol(G') = Pol(G) \cup v,$$
$$V(G') = \{V(G) \cup \{v\} \mid |v|>0\}.$$

- $G + w$ is the addition of the edge to the graph. An edge is formed from the already existing poles of the graph by combining them into a single set. Let $I(w)$ be the set of input poles of an edge w, and $O(w)$ be the set of output poles of w, then the operation is represented as:

$$W(G') = \{W(G) \cup \{w\} \mid |I(w)|>0 \text{ and } |O(w)|>0\}.$$

- $w + p1$ is the addition of the inner pole to the edge. An existing pole belonging to one of the vertexes is added to the edge ($\exists v \in V(G)\ [p1 \in v]$), which is represented as:

$$w' = w \cup \{p1\}.$$

- $w + p2$ is the addition of the outer pole to the edge. An existing pole belonging to the set of outer poles ($p2 \in P(G)$) of the graph is added to the edge:

$$w' = w \cup \{p2\}.$$

- $G + p2$ is the addition of the outer pole to the graph. A pole is added to both the set of outer poles of the graph G and the set of all poles:

$$Pol(G') = Pol(G) \cup \{p2\},$$
$$P(G') = P(G) \cup \{p2\}.$$

The following operations remove elements from an *HP*-graph:

- $v - p1$ is the removal of the inner pole from the node. When a pole is removed from a vertex, all its occurrences in the edges are cut off and it is removed from the set of all poles of the graph:

$$\forall w \in W(G)\ [w = \{w \setminus \{p1\} \mid \{p1\} \in w\}],$$
$$Pol(G') = Pol(G) \setminus \{p1\},$$
$$v' = \{v \setminus \{p1\} \mid |v|>1\}.$$

- $G - v$ is the removal of the node from the graph. In addition to deleting a vertex, all occurrences of the poles of this vertex in the edges are cut off, and all poles of the vertex are removed from the set of poles of the graph:

$$\forall w \in W(G)\ \forall p \in v\ [w = \{w \setminus \{p\} \mid p \in w\}],$$
$$Pol(G') = Pol(G) \setminus \{v\},$$
$$V(G') = V(G) \setminus \{v\}.$$

- $G - w$ is the removal of the edge from the graph. Performing this operation only removes an edge from the set of all edges of the graph, leaving the external poles and vertex poles unchanged:

$$W(G') = W(G) \setminus \{w\}.$$

- $w - p1$ is the removal of the inner pole from the edge. A pole is removed only from an edge w, without changing the set of all poles of the graph and the set of poles of the vertex to which it belongs, but if the resulting edge w' does not contain at least one input and one output pole, then the edge is removed:

$$w' = w \setminus \{p1\},$$
$$W(G') = \{W(G) \setminus \{w\} \mid |I(w)| = 0 \text{ or } |O(w)| = 0\}.$$

- $w - p2$ is the removal of the external pole from the edge, which is equal to the previous operation.

- $G - p2$ is the removal of the outer pole from the graph, which also includes removing this pole from all edges that contain this pole:

$$\forall w \in W(G)\ [w = \{w \setminus \{p2\} \mid \{p2\} \in w\}],$$
$$P(G') = P(G) \setminus \{p2\},$$
$$Pol(G') = Pol(G) \setminus \{p2\}.$$

3.2 Operations of Decomposition

A hypergraph with poles allows vertices and edges to be decomposed during the decryption operation. This feature makes multilevel representation possible. This possibility is achieved by correctly correlating the poles of the source and received graph which is done by implementing a mapping function.

The mapping f: $v \rightarrow P$, which is a decoding function for a vertex v, must be concordant with the sets $I(v)$ and $O(v)$, so that $\forall p \in I(v)$: $[f(p) \in I(G)]$, $\forall r \in O(v)$: $[f(r) \in O(G)]$. Thus, the mapping of the pole $p \in v$ to the next level of the hierarchy is represented as $f(p) = \pi$, where $\pi \in P(G)$, which means that a pole p becomes the external pole π for a resulting graph.

Fig. 2 illustrates decomposition of the vertex v_3 by a new *HP*-graph.

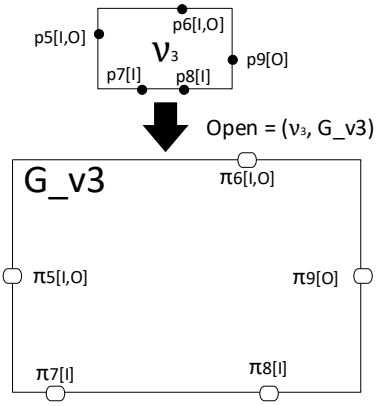

Fig. 2. Example of vertex decomposition by a new HP-graph

An edge can be decomposed similarly but with the help of mapping f: $w \rightarrow P$ which also must be concordant with sets of input ($I(w)$) and output ($O(w)$) poles, so that $\forall p \in I(w)$: $[f(p) \in I(G)]$, $\forall r \in O(w)$: $[f(r) \in O(G)]$. Thus, the mapping of the pole $p \in w$ to the next level of the hierarchy is also represented as $f(p) = \pi$, where $\pi \in P(G)$. Example of decomposition of the edge w_6 is demonstrated in Fig. 3.

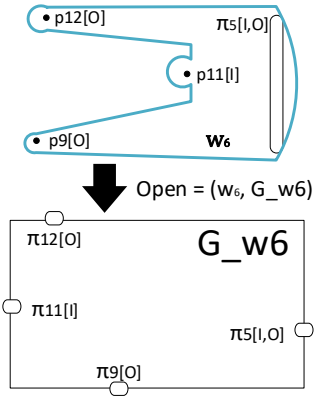

Fig. 3. Example of edge decomposition by a new HP-graph

As is seen, the decomposition of edges and vertices is almost equal, therefore, it is possible to define a common decryption algorithm for these structures.

To do it, let us define a set of structures $Str = V \cup W$. Hence, $str \in Str$ is a structure which can be either a vertex or an edge. The algorithm of the structure decomposition by a new *HP*-graph can be described as follows (Listing 1):

```
Procedure DecomposeStructure:
   G = new HPGraph();
   foreach p∈str:
      if (p∈I(str)):
         I(G) = I(G) ∪ p;
      if (p∈O(str)):
      O(G) = O(G) ∪ p;
   Open_str = Open_str ∪ (str, G)
```

Listing 1. Pseudocode of the algorithm of structure decomposition by a new HP-graph

As is seen from the algorithm, for every structure *str* several decoding operations can be defined. Generally, they can be presented as $Open_{str} \subset str \times G_{all}$, where G_{all} is the set of all *HP*-graphs determined on the set *Pol*.

An edge of an *HP*-graph can also be decrypted by ordinary links between the poles. To implement this operation, it is necessary to define the set $E_w = \{e_1,\ldots,e_n\} \subset I(w) \times O(w)$ for each edge $w \in W$, so that every link ($e \in E_w$) is represented by a pair (p, r) provided that $p \in I(w)$, $r \in O(w)$. Thus, the decoding of the edge $w \in W$ can be represented by the mapping function f: $w \rightarrow E_w$, which replaces the hyperedge with normal connections between the input and output poles.

Fig. 4 illustrates the example of hyperedge decoding by ordinary links. As $E_w \subset I(w) \times O(w)$, some input and output poles can be unconnected such as poles $p9[O]$ and $p11[I]$ in fig. 4.

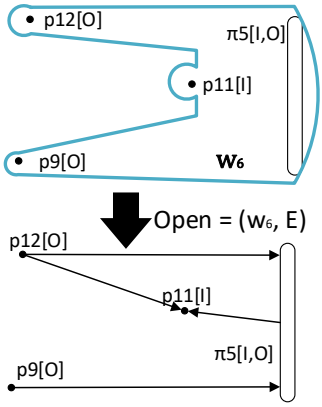

Fig. 4. Example of edge decomposition by ordinary links

3.3 Operations of Transformation

Many different approaches are used to transform visual models, but from the point of view of some scientists and developers [8], [29] the most promising one is the algebraic approach [30], which allows parsing graphs and checking graph models for consistency. This approach and its modifications are implemented in such tools as *MetaLanguage* [8], *AGG* [31] and *VIATRA* [32]. It is worth mentioning that there are also toolsets, such as *ATL* [14], that implement technologies from other areas of software engineering, but most of them have considerable restrictions.

To determine transformation operations, it is necessary to give a definition to a subgraph of a *HP*-graph. A *subgraph* of the HP graph $G = (P, V, W)$ is an *HP*-graph $G' = (P', V', W')$ that is a part of

the graph $G(P' \subset P \,\&\, (\forall v' \in V' \quad \exists v \in V: [v' \subset v]) \,\&\, W' \subset W)$ and fulfills the condition $Open' \subset Open$. The subgraph must also meet the condition (1) to make transformation operations possible.

$$(\forall v \in V' \setminus V'_{partial}\,[p \in v] \,\&\, \exists w \in W[p \in w]) \rightarrow w \in W \quad (1)$$

The set $V'_{partial}$ is a set of the *incomplete vertices* in the graph, where $V'_{partial} \subset V'$.

A subgraph can contain vertices called *incomplete* whose sets of poles can only be a part of the sets of poles of the vertices of the original graph.

To perform a transformation, it is needed to select the source and the target graph and set production rules that describe the transformation. A production rule is represented as $p = (G_L, G_R)$, where G_L is a pattern-graph and G_R is a replacement graph. Nevertheless, there is a *restriction* which is represented in (1) and must be satisfied to perform a transformation. It can be explained by the fact that it is unknown how certain hyperedges should change during the transformation while this restriction obliges to redefine all edges which are incident to poles involved in the transformation.

To display all hyperedges, all the poles that are included in them must be displayed, so it is needed to add such auxiliary (incomplete) vertexes that store only those poles that belong to the displayed hyperedges.

An algorithm for the transformation can be divided into two functions. The first one removes a subgraph isomorphic to the pattern and the second one adds a replacement graph to the original graph.

The first step can be described as follows (listing 2):

```
Procedure Function DeleteGraph(HostG, GL):
   G′ = Find_Isomorphic_Subgraph(HostG, GL);
partials = {}
foreach w′∈W(G′):
W(HostG) = W(HostG) \ {w′};
foreach v′∈V(G′):
   if (v′∈V(HostG)):
      V(HostG) = V(HostG) \ {v′};
   else:
      partials = partials ∪ {v′};
foreach p′∈P(G′):
   if (¬∃w∈W(HostG)[p′∈w]):
      P(HostG) = P(HostG)\p′;
return partials;
```

Listing 2. Pseudocode of the algorithm that removes a subgraph isomorphic to the pattern

The second step of the algorithm will be following (listing 3):

```
Procedure AddGraph(HostG, GR, partials):
foreach p∈P(GR):
   if p∉P(HostG):
      P(HostG) = P(HostG) ∪ {p};
foreach v∈V(GR):
   if (v∉Partials):
      V(HostG) = V(HostG) ∪ {v};
foreach w∈W(GR):
   W(HostG) = W(HostG) ∪ {w};
```

Листинг 3. Псевдокод алгоритма добавления графа замены в исходный граф

These algorithms can be repeated several times as the set of transformation rules may not be limited to just one rule.

4. Justification of expressive power of formalism

It is possible to justify the transcending expressive power of the *HP*-graph by proving that the graph formalisms generally used for building and visualizing models can be represented as an *HP*-graph. Previously, it was mentioned that oriented graphs, hypergraphs, *hi*-graphs, meta-graphs and *P*-graphs are most frequently used for such purposes. Table 2 describes formulas which represent these graph structures as an *HP*-graph.

Table 2. Representation of graphs as an HP-graph

Graph model	**Representation in the HP-graph $G' = (P', V', W')$**
Oriented Graph $G = (V, E)$	$V = P' = V'$, where $\forall v' \in V'$: $[\|v'\| = 1]$ $E = W'$, where $\forall w' \in W'$: $[\|w'\| = 2]$)
Hypergraph $G = (X, E)$	$X = P' = V'$, where $\forall v' \in V'$: $[\|v'\| = 1]$ $E = W'$
Hi-graph $G = (X, E)$	$\{x \mid x \in X \ \& \ \|x\| = 1\} = P' = V'$, where $\forall v' \in V'$: $[\|v'\| = 1]$ $E \cup \{x \mid x \in X \ \& \ \|x\| > 1\} = W'$
Metagraph $G = (V, MV, E)$	$V = P' = V'$, where $\forall v' \in V'$: $[\|v'\| = 1]$ $E \cup MV = W'$
P-graph $G = (P, V, W)$	$P = P'$ $V = V'$ $W = W'$, where $\forall w' \in W'$: $[\|w'\| = 2]$

From the table it can be concluded that the *HP*-graph has more expressive power than the previously described graph models. These graph models are special cases of the *HP*-graph; thus, the *HP*-graph is a generalization of all of these graph formalisms.

5. Conclusion

The definition of the mathematical apparatus underlying the visual model editor was given above, including a detailed description of the graph structure itself, as well as the operations that can be performed on it. For the selected graph formalism, algorithms for decoding vertices and edges, as well as algorithms for performing transformations, were described.

The *HP*-graph combines expressive possibilities of various types of graphs, therefore, algorithms that are designed for these types of graphs (particularly model transformation algorithms [33], [34]) can also be implemented for *HP*-graphs. The time complexity of model transformation algorithms can be reduced. The paper proves that *HP*-graph allows the creation of a flexible visual model editor based on this graph formalism for a DSM platform. Representing both vertices and links as sets of poles simplifies the object model of DSM editor and visual model editing algorithms.

It is planned to develop a program that will demonstrate the practical significance of the selected graph formalism.

References / Список литературы

[1]. Koznov D.V. Basics of visual modeling. Knowledge laboratory, Internet University of information technologies; BINOM, 2008, 280 p. (in Russian) / Кознов Д. В. Основы визуального моделирования. Лаборатория знаний, Интернет-университет информационных технологий; БИНОМ, 2008 г., 280 стр.

[2]. Fugetta A.A classification of CASE technology. Computer, vol. 26, no. 12, 1993, pp. 25-38.

[3]. Ward M.P. Language Oriented Programming. Software – Concepts & Tools, vol. 15, no. 4, 1994, pp. 47-161.

[4]. Sukhov A.O. Comparison of visual object-oriented language development systems. Mathematics of software systems: Intercollegiate collection of scientific articles, 2012, issue 9, pp. 84-111 (in Russian) / Сухов А.О. Сравнение систем разработки визуальных предметно-ориентированных языков.

Математика программных систем: межвузовский сборник научных статей, 2012, вып. 9, стр. 84-111.

[5]. Kelly S., Lyytinen K., Rossi. M. MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE Environment. Lecture Notes in Computer Science, vol. 1080, 1996, pp. 1-21.

[6]. Lyadova L.N., Sukhov A.O., Zamyatina E.B. An Integration of Modeling Systems Based on DSM-Platform. In Proc. of the 18th International Conference on Computers, 2014, pp. 421-425.

[7]. Sukhov A.O., Lyadova L.N. MetaLanguage: a Tool for Creating Visual Domain-Specific Modeling Languages. In Proc. of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering, 2012, pp. 42-53.

[8]. Sukhov A.O. Development of tools for creating visual subject-oriented languages. PhD thesis, Moscow, 2013, 256 p. (in Russian) Сухов А.О. Разработка инструментальных средств создания визуальных предметно-ориентированных языков. Диссертация на соискание ученой степени кандидата физ.-мат. наук, М., 2013. 256 стр.

[9]. Sukhov A.O., Lyadova L.N., Poryazov S.A. Hypergraphs with poles as the basis for creating visual language editors. Mathematics of software systems, no. 15, 2018, pp. 97-104 (in Russian) / Сухов А.О. Лядова Л.Н., Порязов С.А. Гиперграфы с полюсами как основа для создания редакторов визуальных языков. Математика программных систем, вып. 15, 2018, стр. 97-104.

[10]. Microsoft. Visual Studio Docs. Overview of Domain-Specific Language Tools. Available at: https://docs.microsoft.com/en-us/visualstudio/modeling/overview-of-domain-specific-language-tools?view=vs-2019, accessed 10.01.2020.

[11]. Vujovic V., Maksimovic M., Perisic B. Comparative analysis of DSM Graphical Editor frameworks: Graphiti vs. Sirius. In Proc. of the 23rd International Electrotechnical and Computer Science Conference ERK, 2014, pp. 7-10.

[12]. Pavlinov A.A. A set of tools for developing problem-oriented visual languages. Bulletin of Saint Petersburg University, vol. 10, no. 1, 2007, pp. 86-96 (in Russian) / Павлинов А.А. Комплекс средств разработки проблемно-ориентированных визуальных языков. Вестник Санкт-Петербургского университета, том 10, вып. 1, 2007, стр. 86-96.

[13]. Terekhov A.N. Architecture of the visual modeling environment QReal. System programming, no. 4, 2009, pp. 172-197 (in Russian) / Терехов А.Н. Архитектура среды визуального моделирования QReal. Системное программирование, вып. 4, 2009, стр. 172-197.

[14]. Bezivin J., Jouault F., Touzet D. An Introduction to the ATLAS Model Management Architecture, Laboratoire d'Informatique de Nantes-Atlantique, Research Report No. 05.01, 2005, 24 p.

[15]. Jeong K., Wu L., Hong J. IDEF method-based simulation model design and development. Journal of Industrial Engineering and Management, vol. 2, no. 2, 2009, pp. 337-359.

[16]. Serifi V., Dasic P., Jecmenica R., Labovic D. Functional and Information Modeling of Production Using IDEF Methods. Strojniški vestnik – Journal of Mechanical Engineering, vol. 55, no. 2, 2009, pp. 131-140.

[17]. Imran S., Foping F., Feehan J., Dokas I. Domain Specific Modeling Language for Early Warning System: Using IDEF0 for Domain Analysis. International Journal of Computer Science Issues, 2010, vol. 7, no. 5, pp. 10-17.

[18]. OMG. Unified Modeling Language Specification. Available at: https://www.omg.org/spec/UML/2.5.1/PDF, accessed 15.02.2020.

[19]. James P., Knapp A., Mossakowski T., Roggenbach M. Designing Domain Specific Languages – A Craftsman's Approach for the Railway Domain Using CASL. Lecture Notes in Computer Science, vol. 7841, 2012, pp. 178-194.

[20]. Wise R., Brimhall E. A Systems Engineering Approach to the Development of a Domain-Specific Language for Functional Reference Architectures. In Proc. of the 16th Annual Conference on Systems Engineering Research, 2019, pp. 241-254.

[21]. Velter M. MD*/DSL Best Practices Update March 2011. Version 2.0. Available at: http://www.voelter.de/data/pub/DSLBestPractices-2011Update.pdf, accessed 20.03.2020.

[22]. Struchkov I.V. A formalism for describing software systems and computational processes for cyclic parallel processing of real time data. Information and control systems, issue 2, 2006, pp.8-13 (in Russian) / Стручков И.В. Формализм для описания программных систем и вычислительных процессов циклической параллельной обработки данных реального времени. Информационно-управляющие системы, вып. 2, 2006, стр. 8-13.

[23]. Courcelle B. Recognizable Sets of Graphs, Hypergraphs and Relational Structures: A Survey. Lecture Notes in Computer Science, vol. 3340, 2005, pp. 1-11.

[24]. Grosu R., Stefanescu Gh., Broy M. Visual Formalisms Revisited. In Proc. of the 1998 International Conference on Application of Concurrency to System Design, 1998, pp. 41-51.
[25]. Power J., Tourlas K. Abstraction in Reasoning about Higraph-Based Systems. Lecture Notes in Computer Science, vol. 2620, 2003, pp. 392-408.
[26]. Basu A., Blanning R. Graphs, Hypergraphs, and Metagraphs. In Metagraphs and Their Applications. Integrated Series in Information Systems, vol. 15, 2007, pp. 1-12.
[27]. Mikov A.I. Performance evaluation: textbook, Kuban State University, Krasnodar, 2013, 99 p.
[28]. Filatov D.Ju., Lyadova L.N. Development of P-graph based visual model editor. In Proc. of the VIII International Scientific and Technical Conference on Information Systems Development Technologies, 2017, pp. 113-118 (in Russian) / Филатов. Д.Ю., Лядова Л.Н. Разработка редактора визуальных моделей, основанного на P-графах. Материалы VIII Международной научно-технической конференции «Технологии разработки информационных систем», 2017, стр. 113-118.
[29]. Parra F. Dean T. Survey of Graph Rewriting applied to Model Transformations. In Proc. of the 2nd International Conference on Model-Driven Engineering and Software Development, 2014, pp. 431-441.
[30]. Ehrig H., Ehrig K., Prange U., Taentzer G. Fundamentals of Algebraic Graph Transformation, Springer, 2006, 388 p.
[31]. AGG. The Homebase. A brief Description of AGG. Available at: https://www.user.tu-berlin.de//o.runge/agg/agg-docu.html, accessed 12.03.2020.
[32]. Eclipse Modeling Project. Eclipse VIATRA. Available at: https://www.eclipse.org/viatra/, accessed 12.03.2020.
[33]. Seriy A.P., Lyadova L.N. An Approach to Graph Matching in the Component of Model Transformations. In Proc. of the 7th Spring/Summer Young Researchers' Colloquium on Software Engineering, 2013, pp. 41-46.
[34]. Sukhov A., Lyadova L.N. Horizontal Transformations of Visual Models in MetaLanguage System. In Proc. of the 7th Spring/Summer Young Researchers' Colloquium on Software Engineering, 2013, pp. 31-40.

Information about authors Информация об авторах

Nikolai Mikhailovich SUVOROV – undergraduate student of the HSE (Perm). His research interests include language-oriented programming, modeling, language toolkits.

Николай Михайлович СУВОРОВ – студент бакалавриата НИУ ВШЭ (Пермь). Научные интересы включают языково-ориентированное программирование, моделирование, языковые инструментарии.

Lyudmila Nickolaevna LYADOVA – Candidate of Physical and Mathematical Sciences, associate professor of the Department of Information Technology in Business of the HSE (Perm). Research interests: modeling languages, modeling tools, domain specific modeling, language toolkits, semantic modeling.

Людмила Николаевна ЛЯДОВА – кандидат физико-математических наук, доцент, доцент кафедры информационных технологий в бизнесе НИУ ВШЭ (Пермь). Сфера научных интересов: языки моделирования, средства моделирования, предметно-ориентированное моделирование, языковые инструментарии, семантическое моделирование.

DOI: 10.15514/ISPRAS-2020-32(2)-13

Платформа автоматического фаззинга программного интерфейса приложений

[1]*С.С. Саргсян, ORCID: 0000-0002-8831-4965 <sevaksargsyan@ispras.ru>*
[1]*В.Г. Варданян, ORCID: 0000-0002-4899-2999 <vaag@ispras.ru>*
[1]*Дж.А. Акопян, ORCID: 0000-0002-4094-2727 <jivan@ispras.ru>*
[1]*А.М. Агабалян, ORCID: 0000-0003-2331-6692 <anna.aghabalyan@ispras.ru>*
[1]*М.С. Меграбян, ORCID: 0000-0001-9846-3414 <matos@ispras.ru>*
[2]*Ш.Ф. Курмангалеев, ORCID: 0000-0002-0558-2850 <kursh@ispras.ru>*
[2]*А.Ю. Герасимов, ORCID: 0000-0001-9964-5850 <agerasimov@ispras.ru>*
[2]*М.К. Ермаков, ORCID: 0000-0002-0483-6097 <mermakov@ispras.ru>*
[2]*С.П. Вартанов, ORCID: 0000-0003-3786-2248 <svartanov@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. Рандомизированное тестирование приложений (фаззинг, фаззинг-тестирование) является одним из широко используемых методов поиска ошибок. Цель фаззинг-тестирования – определить стабильность приложений при обработке псевдослучайно сгенерированных входных данных. В ходе тестирования приложение запускается на множестве произвольных входных данных, которые могут быть недействительными/неожиданными. Современное программное обеспечение часто предоставляет программный интерфейс (Application programming interface) для расширения возможностей программы. Это еще больше усложняет тестирование программного обеспечения, поскольку становится необходимым учитывать все возможные сценарии использования предоставленных интерфейсных функций. Применение фаззинга для генерации разных сценариев использования программного интерфейса приложения и соответствующих входных данных позволяет эффективным образом выявить ошибки в реализации функций программного интерфейса. В данной статье описывается новый метод фаззинг-тестирования для Android приложений и библиотек, написанных на языке Java. Разработанный инструмент фаззинг-тестирования выявил 15 уникальных дефектов, приводящих к аварийному завершению приложения SmartThings, разработанного компанией Samsung.

Ключевые слова: интернет вещей; фаззинг; генерация вызовов API

Для цитирования: Саргсян С.С., Варданян В.Г., Акопян Дж.А., Агабалян А.М., Меграбян М.С., Курмангалеев Ш.Ф., Герасимов А.Ю., Ермаков М.К., Вартанов С.П. Платформа автоматического фаззинга программного интерфейса приложений. Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 161-174. DOI: 10.15514/ISPRAS–2020–32(2)–13

Благодарности: Работа поддержана компанией Samsung-Electronics.

Automatic API fuzzing framework

[1]*S.S. Sargsyan, ORCID: 0000-0002-8831-4965 <sevaksargsyan@ispras.ru>*
[1]*V.G. Vardanyan, ORCID: 0000-0002-4899-2999 <vaag@ispras.ru>*
[1]*J.A. Hakobyan, ORCID: 0000-0002-4094-2727 <jivan@ispras.ru>*
[1]*A.M. Aghabalyan, ORCID: 0000-0003-2331-6692 <anna.aghabalyan@ispras.ru>*
[1] *M.S. Mehrabyan, ORCID: 0000-0001-9846-3414 <matos@ispras.ru>*
[2]*Sh.F. Kurmangaleev, ORCID: 0000-0002-0558-2850 <kursh@ispras.ru>*
[2]*A.Yu. Gerasimov, ORCID: 0000-0001-9964-5850 <agerasimov@ispras.ru>*
[2]*M.K. Ermakov, ORCID: 0000-0002-0483-6097 <mermakov@ispras.ru>*
[2]*S.P. Vartanov, ORCID: 0000-0003-3786-2248 <svartanov@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. Randomized testing (fuzzing) is a well-known approach for finding bugs in programs. Fuzzing is typically performed during the finishing stage of quality assurance in order to check the stability of the target program in the face of malformed or unexpected input data. Modern software more than often provides an API for extending its functionality by third-party developers; since an API is an entry point to software internals, its functionality and usage scenarios must be tested as well. Thorough API testing must involve checking a large number of possible scenarios and it is fairly obvious that fuzzing can be applied to this task by generating usage scenarios in an automatic randomized way—which brings us to the concept of API fuzzing. In this paper we describe an automatic approach to randomized testing of API libraries for Android/desktop Java. Proposed method is able to change the sequence of called API functions in order to discover new execution paths. It consists of two basic stages. In the first stage the arguments of currently called API functions are mutated. When mutation of called API functions arguments can't find new execution path the tool switches to the second stage. In the second stage current sequence of API functions calls is mutated. Mutation can add new API functions calls or remove some of them. After API calls sequence mutation, the tool switches back to the first stage. Switches between the first and the second stages are continued during whole process of fuzzing. During the experimental setup developed method of randomized testing were able to find 15 crashes in SmartThings application developed by Samsung.

Keywords: internet of things; fuzzing; API call generation

For citation: Sargsyan S.S., Vardanyan V.G., Hakobyan J.A., Aghabalyan A.M., Mehrabyan M.S., Kurmangaleev Sh.F., Gerasimov A.Yu., Ermakov M.K., Vartanov S.P. Automatic API fuzzing framework. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 161-174 (in Russian). DOI: 10.15514/ISPRAS–2020–32(2)–13

Acknowledgements: The work has been done with support of Samsung Electronics.

1. Введение

Контроль качества программного обеспечения является одной из наиболее важных задач при его разработке. Для решения этой проблемы компании внедряют в жизненный цикл разработки программного обеспечения различные инструменты анализа программ [1]. Спектр инструментов достаточно широк и может включать в себя статический и динамический анализ кода программ. Фаззинг [2] является одним из наиболее распространенных и эффективных методов динамического анализа. Метод применялся ещё в середине 1950-х годов, но первые был описан в работе посвященной оценке надёжности Unix утилит [3]. Эффективность предложенного метода подтверждается множеством дефектов, найденных на разных утилитах из инструментария UNIX [4]. Методы и подходы фаззинга постепенно совершенствуются, и в данный момент существует множество инструментов, предназначенных для фаззинг-тестирования. AFL [5] использует

генетический алгоритм выборки входных данных, что позволяет эффективно увеличивать покрытие кода. В работе [6] описывается генерация структурированных данных, заданных в форме BNF, для фаззинга компиляторов, интерпретаторов и трансляторов. Инструмент [7] производит направленный фаззинг на основе динамической инструментации целевой программы. В работе [8] фаззинг совмещается с динамическим символьным исполнением программы с целю увеличения покрытия кода. Проект libFuzzer [9] используется для фаззинга библиотечных функций написанных на языке C/C++. Разработчик должен дописать код функции-обёртки для фаззинга конкретной функции из библиотеки. Программные инструменты Trinity [10] и syzkaller [11] предназначены для фаззинга системных вызовов операционной системы Linux [12]. Эти инструменты генерируют множество маленьких программ, содержащих системные вызовы, и запускают их в окружении целевой операционной системы. При запуске производится мониторинг состояний системы в целях обнаружения сбоев.

Для фаззинга библиотечных/интерфейсных функций существующие инструменты используют либо аннотации этих функций, либо адаптеры, которые обеспечивают их вызов. Это приводит к невозможности полностью автоматического их применения для новых библиотечных или интерфейсных функций. Современное программное обеспечение часто предоставляет большое число интерфейсных функций, которые используются для реализации подключаемых модулей и позволяют расширять функциональные возможности программы. Предоставляемые интерфейсные функции должны иметь конкретную спецификацию и сценарии использования. Система предоставляющая интерфейсные функций должна уметь обрабатывать все недействительные сценарии их использования. В общем случае задача может быть неразрешимой, поскольку количество сценариев возрастает экспоненциально от количества предоставляемых функций. Применение фаззинга к таким системам может эффективным образом выявлять недействительные сценарии использования предоставляемой функциональности.

В данной работе предлагается новый подход для фаззинга интерфейсных функций из библиотек языка Java (JAR файлы [13]). Предлагаемый метод имеет несколько преимуществ:

1. Полностью автоматически восстанавливаются аннотации функций и классов из JAR файла, что позволяет при изменении версии библиотеки (добавление/удаление интерфейсных функций) заново генерировать аннотации.
2. Создается генератор последовательности вызовов интерфейсных функций и их аргументов, который может отправлять сгенерированные данные на выбранные устройства (например, мобильные телефоны). В том числе он может производить параллельный запуск функций на множестве устройств с синхронизацией результатов.
3. Обеспечивается поддержка шаблонов начальных сценариев использования интерфейсных функций (например, инициализация и очищение ресурсов до или после основных вызовов интерфейсных функций). Это позволяет эффективно увеличивать покрытие кода и снижать количество неправильных сценариев, увеличивая эффективность применения фаззера.

Предлагаемый подход был реализован в рамках инструмента ISP-Fuzzer [14], разработанного в Институте системного программирования им. В. П. Иванникова РАН и протестированного на библиотеке PluginBase, которая входит в состав платформы SmartThings [15], созданной компанией Samsung [16]. В результате экспериментального запуска разработанный инструмент смог выявить 15 уникальных дефектов, приводящих к отказу системы, что демонстрирует эффективность предлагаемого решения. Полученные результаты были подтверждены компанией Samsung.

2. Высокоуровневое описание платформы

На рис. 1 приводится общая схема предлагаемой платформы. Серверная часть инструмента производит генерацию данных (последовательность вызовов интерфейсных функций с соответствующими аргументами), которые будут выполнены на клиентской стороне. Серверная часть может одновременно работать с множеством клиентских приложений и синхронизовать полученные результаты. Это позволяет многократно увеличить эффективность фаззинга.

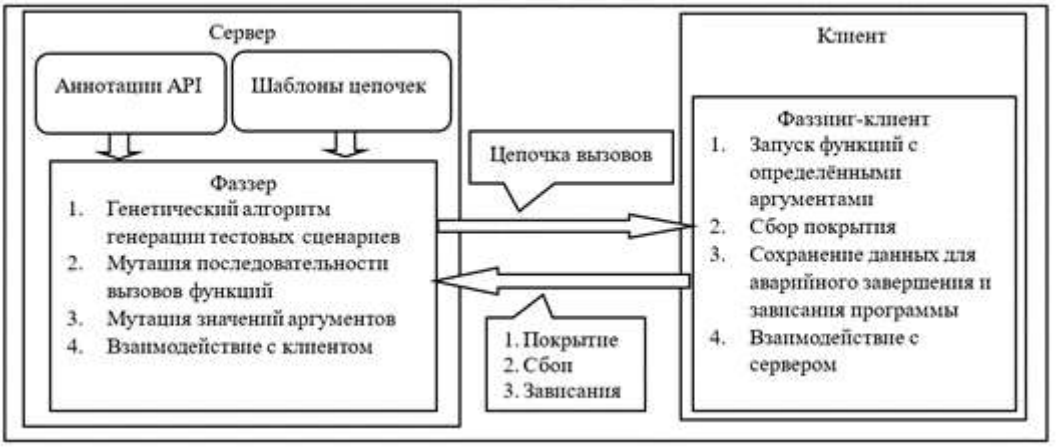

Рис. 1. Общая схема предложенной платформы
Fig. 1. The general scheme of the proposed platform

На основе аннотаций функций, которые передаются инструменту, строятся последовательности (цепочки) вызовов интерфейсных функций. Разработан специальный протокол передачи данных между сервером и клиентами, который позволяет отправлять сгенерированную последовательность вызовов интерфейсных функций и получать необходимое покрытие кода. В зависимости от того, какое покрытие обеспечивает сгенерированная последовательность вызовов, генетический алгоритм подбирает следующую последовательность вызовов и набор аргументов. Последовательности вызовов, которые привели к аварийному завершению или зависанию программы, сохраняются для последующего анализа экспертами.

С помощью конфигурационного файла можно указать шаблоны генерации новых последовательностей вызовов. Например, можно указать интерфейсные функций, которые будут вызваны перед и после каждой сгенерированной последовательности вызовов для реализации специфического протокола использования программного интерфейса.

2.1 Фаззинг интерфейсных функций на примере библиотеки PluginBase

На рис. 2 приводится схема фаззинга интерфейсных функций из библиотеки PluginBase (JAR файл), входящей в состав SmartThings. SmartThings платформа связывает разные устройства интернета вещей и осуществляет управление ими. Управление устройствами производится через интерфейсные функции, входящие в библиотеку PluginBase. Для выполнения фаззинга библиотеки PluginBase были разработаны два инструмента. Первый инструмент генерирует аннотации для классов и методов, входящих в JAR библиотеку. На основе этих аннотаций генерируются последовательности вызовов интерфейсных функций для исполнения на клиентской машине. Если функция получает как аргумент производный объект, система сгенерирует код для создания этого объекта. Построение сложных объектов производится следующим образом:

1. Если есть интерфейсная функция, которая возвращает объект данного типа, производится вызов этой функции.
2. В противном случае вызывается конструктор данного объекта. Если конструктор принимает как аргумент сложный объект, процесс повторяется рекурсивно пока все аргументы создаваемого объекта не будут иметь примитивный тип или тип, сконструированный из примитивных типов.

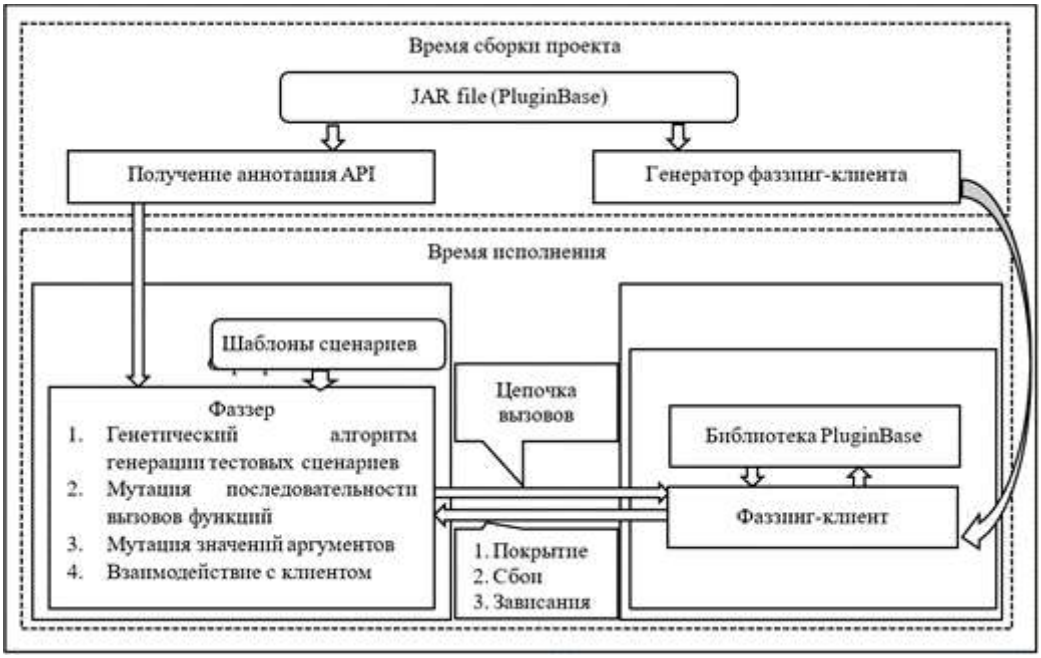

Рис. 2. Схема фаззинга интерфейсных функций PluginBase
Fig. 2. Fuzzing scheme of PluginBase interface functions

Второй инструмент генерирует встраиваемый модуль (плагин) для приложений SmartThings, который получает последовательность вызовов интерфейсных функций от сервера и выполняет их на клиенте (мобильном телефоне). Реализация встраиваемого модуля представляет собой конечный автомат, поддерживающий протокол общения с сервером (получает последовательность команд, отправляет покрытие кода и информацию о падениях/зависаниях приложения). Дополнительно инструментируется код самой библиотеки PluginBase для получения покрытия кода в процессе исполнения цепочек вызовов.

2.2 Генерация вызовов интерфейсных функций

Генерация последовательности вызовов API функций состоит из двух основных этапов. На первом этапе производится вызов всех API функций (разминка – warm-up). Цель данного этапа – сбор начального покрытия кода, который обеспечивает эффективную работу генетического алгоритма. Если этот этап пропущен, добавление вызова новой функции в цепочку вызовов всегда будет увеличивать покрытие кода и генетический алгоритм не сможет отличить эффективные мутации последовательности вызовов и аргументов функций от неэффективных. Кроме этого, на данном этапе всем функциям передаются нулевые указатели на объекты, в целях определения необработанных случаев использования нулевых

указателей (*NullPointerException*). На этапе разминки, как правило, достигается покрытие кода по базовым блокам в 30-40%.

На втором этапе (рис. 3) начинается основной цикл фаззинга (мутация цепочки вызовов функций и их аргументов). Произвольным образом генерируется набор начальных цепочек (мутацию цепочки вызовов можно ограничить через конфигурационный файл шаблонов цепочек). Одна из начальных цепочек вызовов функций загружается для обработки. Производятся мутации всех аргументов функций из цепочки, которые продолжаются до тех пор, пока покрытие кода перестает увеличивается. В этот момент инструмент переходит к мутациям последовательности вызовов функций в цепочке. В процессе этой мутации могут быть добавлены или удалены некоторые вызовы. Если сгенерированная цепочка вызовов увеличивает покрытие кода, она сохраняется для дальнейшей обработки. Чем больше увеличивается покрытие кода, тем дольше будут обрабатываться соответствующая цепочка вызовов и аргументы функций. Наблюдаемый прирост покрытия на второй стадии фаззинга в течение недели составляет ещё 30-40%. Пользователь может управлять процессом мутации, задав в шаблонном файле следующие параметры:

1. последовательности вызовов функций, которые должны быть обработаны первыми;
2. набор мутаций для аргументов функций;
3. набор функций, которые должны быть вызваны до и после каждой цепочки вызовов;
4. зависимости между функциями. В частности, возвращаемое значение одной функций должно быть использовано, как аргумент вызова другой.

Рис. 3. Схема основного цикла фаззинга на серверной машине
Fig. 3. The scheme of the main fuzzing cycle on the server machine

После того, как цепочка вызовов функций и соответствующие аргументы сгенерированы, они передаются следующему компоненту (генератор последовательности команд выполнения – ГПКВ, Execution command sequence generator). ГПКВ получает последовательность команд,

выполнение которых на клиентской стороне обеспечит вызовы интерфейсных функций с соответствующими аргументами.

2.2.1. Мутация данных

Модуль мутации данных получает на вход начальное значение (может быть пустым) и длину требуемой последовательности данных. Возвращает мутированные/сгенерированные данные заданной длинны. В инструменте поддерживается следующий набор мутаций данных:

1. расширение данных: с одинаковым произвольным значением, произвольными значениями, с нулевыми байтами;
2. сокращение данных путем удаления произвольного количества байтов;
3. модификация данных: присвоение последовательности байтов произвольных значения, присвоение последовательности байтов одного и того же произвольного значения, присвоение произвольному интервалу байтов нулевого/произвольного значения;
4. инверсия произвольного бита;
5. генерация строковых данных: повторение строки, создание не валидного списка UTF-8 строк, создание BOM (byte order mark U+FEFF);
6. модификация строковых данных: перевод символов в нижний/верхний регистр;
7. модификация числовых значений: присвоение максимального/минимального значения, применение арифметических операций.

2.2.2. Мутация цепочек вызовов интерфейсных функций

В ходе мутаций цепочек вызовов используется информация о зависимостях интерфейсных функций (автоматически восстановленная или указанная пользователем). Это делается в целях получения семантически «связанных» программ, когда возвращаемое значение одной из функций используется другой. При создании новой цепочки вызовов произвольным образом выбирается некоторая функция *f* из списка доступных интерфейсных функций. Если существует функция, возвращаемое значение которой может быть использовано как аргумент функции *f*, то перед вызовом *f* добавляется вызов этой функции, а возвращаемое значение передаётся как аргумент вызову *f*. Процесс повторяется рекурсивно для добавленных функций до тех пор, пока длина цепочки не превысит заданную границу.

В случае мутаций цепочки производится добавление/удаление некоторой последовательности вызовов, которые не зависят друг от друга. Максимальная длина таких последовательностей задается пользователем. На мутации цепочек вызовов можно повлиять с помощью конфигурационного файла шаблонов. Предоставляется возможность:

1. определить набор функций, которые будут вызваны перед/после каждой цепочки;
2. определить конкретные значения или интервалы значений для аргументов функций;
3. составить список функций, которые могут или не могут присутствовать в цепочках вызовов;
4. задавать зависимости между функциями.

2.2.3. Генерация последовательности команд выполнения

Инструмент имеет два основных назначения:

1. для данной цепочки вызовов вычислять, сколько данных потребуется, чтобы получить конкретные аргументы всех функций;
2. генерировать последовательность байтов, которая будет выполнена/интерпретирована на клиентской машине.

Для выполнения первой своей задачи инструменту необходима цепочка вызовов и аннотаций интерфейсных функций и классов. Для выполнения второй задачи инструменту также необходимо получить конкретные аргументы функций (данные от модуля мутации данных).

Вычисление размера необходимых данных (задача 1) производится рекурсивным обходом составных типов сложного объекта. Размер данных вычисляется суммированием размера составных типов. На этом этапе собирается дополнительная информация о возможных значениях конкретных аргументов. Есть три основных вида таких значений: *MustBeNull* (аргумент всегда должен иметь значение null), *NotNull* (аргумент никогда не должен иметь значение null), *Any* (аргумент может иметь любое значение). Эта информация передается модулю мутаций для генерации корректных данных. Это обеспечивает семантическую корректность вызовов функций. Например, некоторые примитивные типы в Java не могут иметь значение null. Если есть функция с аргументом такого типа, то вызов этой функции (вызов производит клиент фаззинга – интерпретатор последовательности команд) с null аргументом приведет к некорректному поведению программы. На практике такая ситуация возникнуть не может, поскольку компилятор Java не позволит провести компиляцию такого кода.

Табл. 1. Коды операций регистровой машины
Table 1. Codes of operations of the register machine

Описание команды	**Код операций**	**Операнды**
Положить значение в переменную	1-8 (в зависимости от типа)	«переменная», «значение»
Положить UTF значение в переменную	9	«переменная», «UTF строка»
Положить массив в переменную	10	«переменная», «идентификатор типа», «длина», {«значении»}
Положить null в переменную	11	«переменная»
Положить строку в переменную	12	«переменная», «длина», {«значении»}
Положить массив переменных в переменную	13	«переменная», «длина», {«переменные»}
Вызов метода	«method ID»	[«переменная»], «ссылка объекта», {«параметры метода»}
Вызов статического метода	«method ID»	[«переменная»], {«параметры метода»}
Получить значения поля объекта	«get field ID»	«переменная», «переменная, в которой ссылка на объект»
Получить значения статического поля	«get field ID»	«переменная»
Положить значение в поле объекта	«put field ID»	«переменная, в которой ссылка на объект», «переменная, в которой значение»
Положить значение в статическое поле	«put field ID»	«переменная, в которой значение»

Генерация последовательности команд выполнения производится на основе простейшей регистровой машины. Каждая операция (положить значение в переменную, вызвать функцию и так далее) кодируется в протоколе (в табл. 1 приведены коды). Алгоритм проходит по цепочке вызовов функций и получает последовательность команд для регистровой машины.

2.3 Интерпретатор команд

Интерпретатор представляет собой оператор-переключатель, который в зависимости от кода операций выполняет соответствующую операцию. Также имеется буфер переменных, в

котором хранятся необходимые переменные (это, по сути, регистры регистровой машины). В табл. 1 приводятся доступные коды операций. Каждый метод и класс (также поля класса) имеют уникальный идентификатор ID, который позволяет определить нужный метод, класс и поля класса. В таблице этот идентификатор является кодом некоторой команды. На рис. 4 приводится фрагмент кода и соответствующая последовательность команд регистровой машины.

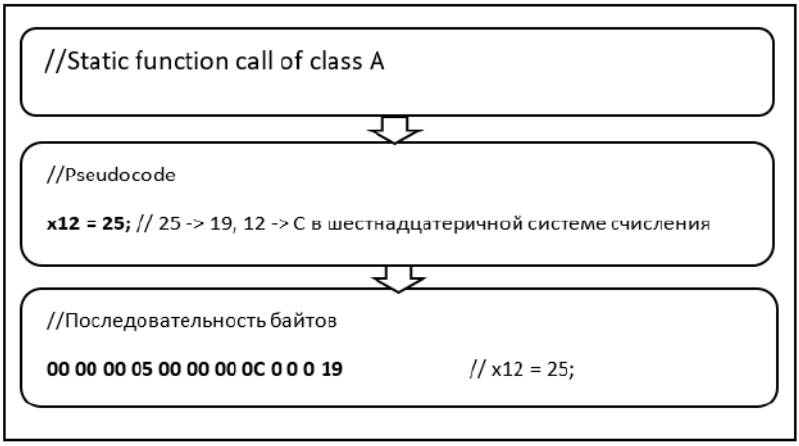

Рис. 4. Пример последовательности команд
Fig. 4. Example sequence of commands

Авторами разработан отдельный инструмент, который из аннотаций JAR-файла генерирует интерпретатор. Это позволяет при обновлении библиотеки автоматически генерировать новый интерпретатор, поддерживающий вызовы добавленных/модифицированных функций.

2.4 Получение покрытия кода

Для получения покрытия кода используется техника, реализованная в инструменте AFL [5]:

1. Каждому базовому блоку целевой библиотеки сопоставляется целое число из интервала [0, 65536). Для сбора покрытия используются две глобальные переменные: карта памяти (map) в размере 65536 элементов и переменная *prevLocation*. Обе переменные инициализированы нулями.
2. При выполнении базового блока X, которому было присвоено целое число bbIndexX (шаг 1), производится следующее изменение в карте памяти:

```
id := bbIndexX ^ prevLocation;
map[id] := map[id] + 1;
prevLocation := bbIndexX >> 1;
```

   Таким образом в карте памяти хранится информация о выполненных базовых блоках.

Такой подход позволяет выявить уникальные пути выполнения (хотя возможны коллизии). Реализация инструментации была выполнена на основе платформы ASM [17].

3. Результаты

В ходе экспериментального запуска на библиотеке PluginBase входящий в состав платформы SmartThings (версия 1.7.9) разработанный инструмент выявил 15 уникальных аварийных завершений программы. Результаты были подтверждены командой тестирования Samsung Electronics R&D. На рис. 5 приводятся некоторые методы, вызовы которых с указанными параметрами приводят к возникновению исключительных ситуаций. Как видно из результатов инструмент может найти не только «простые» ошибки, когда передаются null

объекты. Успешно были найдены исключения выхода индекса за пределы строки и другие типы исключений, определенных в самой платформе SmartThings. Данные результаты были получены в результате параллельного фаззинга на двух устройствах в течении семи дней. На рис. 6 приводится фрагмент кода, демонстрирующий выше указанные ошибки

```
// com.samsung.android.plugins.PluginDataStorageException
JSONConverter.jsonToRcsRep ("{");

// java.lang.StringIndexOutOfBoundsException
ShpConverter.vidToShpSSID ("nohyphen");

// java.lang.NullPointerException
QcPluginDevice.getQcPluginDevice ("foo");

// os.android.RemoteException
AutomationManager.getInstance ().getRegisteredAutomation ("1");
```

Рис. 5. Найденные необработанные исключительные ситуации
Fig. 5. Unhandled exceptions found

```
package com.samsung.android.plugin.tv;
import android.os.Bundle;
import android.view.Window;
import android.view.WindowManager;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.InetSocketAddress;

import com.samsung.android.oneconnect.utils.ShpConverter;
import com.samsung.android.plugins.QcPluginDevice;
import com.samsung.android.plugins.automation.AutomationManager;
import com.samsung.android.sccclient.JSONConverter;

public class MainActivity extends PluginBaseActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        final Window window = getWindow();
        window.setSoftInputMode(
            WindowManager.LayoutParams.SOFT_INPUT_ADJUST_RESIZE |
                WindowManager.LayoutParams.SOFT_INPUT_STATE_HIDDEN);
        getTheme().applyStyle(R.style.AppTheme, true);

        setContentView(R.layout.activity_main);

        System.out.println("JSONConverter.jsonToRcsRep");
        try {
            JSONConverter.jsonToRcsRep("{");
        } catch (Throwable e) {
            e.printStackTrace();
        }

        System.out.println("ShpConverter.vidToShpSSID");
        try {
            ShpConverter.vidToShpSSID("nohyphen");
        } catch (Throwable e) {
            e.printStackTrace();
        }

        System.out.println("QcPluginDevice.getQcPluginDevice");

        try {
            QcPluginDevice.getQcPluginDevice("foo");
        } catch (Throwable e) {
            e.printStackTrace();
        }

        System.out.println("AutomationManager.getRegisteredAutomation");
        try {
            AutomationManager mgr = AutomationManager.getInstance();
            mgr.getRegisteredAutomation("1");
        } catch (Throwable e) {
            e.printStackTrace();
        }
    }
}
```

Рис. 6. Фрагмент кода, демонстрирующий найденные исключения
Fig. 6. Code fragment showing exceptions found

4. Сравнение с аналогичными решениями

EvoSuite [18] является одной из известных платформ, которая автоматически генерирует тесты для Java-классов. Инструмент учитывает покрытие кода, и путем генерации разных тестов может достичь большого/полного покрытия кода. EvoSuite также может генерировать JUnit4 [19] тесты. Данный инструмент имеет некоторые ограничения.

1. Нет возможности генерировать последовательности вызовов функций. Таким образом ошибки, которые воспроизводятся при некоторой последовательности вызовов функций не могут быть найдены.
2. Нельзя указать шаблоны для генерации последовательности вызовов функций. Например, указать корректное создание/освобождение ресурсов перед/после вызовов функций.
3. Если запуск программы производится в специальной среде, EvoSuite тоже не может быть применен. Например, в случае библиотеки PluginBase вызов её методов можно производить только в том случае, когда библиотека загружена через платформу SmartThings (в среде ОС Android [20]).

DSD-Crasher [21] (его предшественник – JCrasher [22]) генерирует тесты для методов Java классов в целях выявления необъявленных исключений. Он также умеет генерировать JUnit тесты. Для достижений этой цели инструмент комбинированно использует динамический и статический анализ. Как и в случае EvoSuite, этот инструмент не может решать поставленную задачу, поскольку имеет те же самые ограничения.

Randoop [23] в отличие EvoSuite и DSD-Crasher может вызывать несколько методов одного класса в одном тесте. Но вызов методов из разных классов не производится. Таким образом генерация последовательности вызовов функций решается частично. Кроме этого, Randoop не учитывает покрытие кода для генераций новых тестов, что является еще одним недостатком.

Таким образом, предложенный авторами метод имеет ряд преимуществ при рандомизированном тестировании интерфейсных функций по сравнению с вышеуказанными инструментами.

Список литературы / References

[1]. SDL. Available at: https://www.microsoft.com/en-us/securityengineering/sdl, accessed 26.11.2019.

[2]. Fuzzingю Available at: https://en.wikipedia.org/wiki/Fuzzing, accessed 26.11.2019.

[3]. Borton P. Miller, Lars Fedriksen. Bryan So. An Empirical Study of the Reliability of UNIX utilities. Communications of the ACM, vol. 33, № 12, 1990, pp. 32-44.

[4]. Unix. Available at: https://en.wikipedia.org/wiki/Unix, accessed 26.11.2019.

[5]. Michael Zelewski. American Fuzzy Lop. 2014. Available at: http://lcamtuf.coredump.cx/afl. Accessed 26.11.2019.

[6]. Sevak Sargsyan, Shamil Kurmangaleev, Matevos Mehrabyan, Maksim Mishechkin, Tsolak Ghukasyan, Sergey Asryan. Grammar-Based Fuzzing. In Proc. of the *2018 Ivannikov Memorial Workshop (IVMEM)*, Yerevan, Armenia, 2018, pp. 32-35, doi: 10.1109/IVMEM.2018.00013.

[7]. Sevak Sargsyan, Shamil Kurmangaleev, Jivan Hakobyan, Matevos Mehrabyan, Sergey Asryan, Hovhannes Movsisyan. Directed Fuzzing Based on Program Dynamic Instrumentation. In Proc. of the 2019 International Conference on Engineering Technologies and Computer Science (EnT), Moscow, Russia, 2019, pp. 30-33, doi: 10.1109/EnT.2019.00011.

[8]. 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.

[9]. libFuzzer. Available at: https://llvm.org/docs/LibFuzzer.html, accessed 26.11.2019.

[10]. Trinity: Linux system call fuzzer. Available at: https://github.com/kernelslacker/trinity, accessed 26.11.2019.

[11]. syzkaller: Linux syscall fuzzer. Available at: https://github.com/google/syzkaller syzkaller, accessed 26.11.2019.
[12]. Linux. Available at: https://www.linux.org/, accessed 26.11.2019.
[13]. JAR. Available at: https://en.wikipedia.org/wiki/JAR_(file_format), accessed 26.11.2019.
[14]. Sevak Sargsyan, Jivan Hakobyan, Matevos Mehrabyan, Maxim Mishechkin, Vitaliy Akozin, Shamil Kurmangaleev. ISP-Fuzzer: Extendable Fuzzing Framework. In Proc. of the 2019 Ivannikov Memorial Workshop (IVMEM), Velikiy Novgorod, Russia, 2019, pp. 68-71, doi: 10.1109/IVMEM.2019.00017.
[15]. SmartThings. Available at: https://www.samsung.com/global/galaxy/apps/smartthings/, accessed 26.11.2019.
[16]. Samsung. Available at: https://www.samsung.com, accessed 26.11.2019.
[17]. Bruneton E., Lenglet R., Coupaye T. ASM: A code manipulation tool to implement adaptable systems. In Proc. of the ASF (ACM SIGOPS France) Journées Composants 2002: Syst`emes `a composants adaptables et extensibles (Adaptable and Extensible Component Systems), Grenoble, France, 2002.
[18]. Gordon Fraser, Andrea Arcuri. EvoSuite: automatic test suite generation for object-oriented software. In Proc. of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, Szeged, Hungary, 2011, pp. 416-419.
[19]. JUnit4. Available at: https://junit.org/junit4/, accessed 26.11.2019.
[20]. Android. Available at: https://www.android.com/, accessed 26.11.2019.
[21]. Christoph Csallner, Yannis Smaragdakis, Tao Xie. DSD-Crasher: A hybrid analysis tool for bug finding. ACM Transactions on Software Engineering and Methodology, vol.17, issue 2, 2008, Article No. 8.
[22]. Christoph Csallner, Yannis Smaragdakis. JCrasher: an automatic robustness tester for Java. Software – Practice & Experience, vol. 34, issue 11, 2004, pp. 1025 - 1050.
[23]. Carlos Pacheco, Michael D. Ernst. Randoop: feedback-directed random testing for Java. In Proc.of the OOPSLA '07 Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion, Montreal, Quebec, Canada, 2007, pp. 815-816.

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.

Ваагн Геворгович ВАРДАНЯН – научный сотрудник, преподаватель, кандидат технических наук. Сфера научных интересов: анализ программ, динамический анализ кода, фаззинг.

Vahagn Gevorgovich VARDANYAN – researcher, lecturer, Ph.D in technical 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.

Анна Мартиросовна АГАБАЛЯН – научный сотрудник, магистр. Сфера научных интересов: анализ программ, динамический анализ кода, фаззинг.

Anna Martirosovna AGHABALYAN – 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.

Шамиль Фаимович КУРМАНГАЛЕЕВ – 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.

Александр Юрьевич Герасимов – кандидат физико-математических наук. Сфера научных интересов: автоматический анализ программ, статический анализ программ, динамический анализ программ, комбинированные методы анализа программ, управление научными исследованиями и разработками, жизненный цикл разработки безопасного ПО.

Alexander Yurievich GERASIMOV – PhD in Computer Sciences. Research interests: automatic program analysis, static program analysis, dynamic program analysis, combined methods for program analysis, R&D management, SSDLC.

Михаил Кириллович ЕРМАКОВ – кандидат технических наук. Сфера научных интересов: динамический анализ кода, фаззинг, символьное исполнение, статический анализ кода.

Mikhail Kirilovich ERMAKOV – Candidate of Technical Sciences. Research interests: dynamic program analysis, fuzzing. Symbolic execution, static program analysis.

Сергей Павлович ВАРТАНОВ – сфера научных интересов: символьное исполнение, динамический и статический анализ программ и SMT-решатели.

Sergey Pavlovitch VARTANOV – research interests: symbolic execution, dynamic and static analysis, and SMT solvers.

DOI: 10.15514/ISPRAS-2020-32(2)-14

Analysis of Russian software supporting onboard systems development lifecycle in context of import substitution policy

N.K. Gorelits, ORCID: 0000-0003-0974-7146 <nkgorelits@2100.gosniias.ru>
A.S. Gukova, ORCID: 0000-0002-8323-5993 <asgukova@2100.gosniias.ru>
D.V. Krasnoschekov, ORCID: 0000-0003-2254-2719 <dvkrasnoshekov@2100.gosniias.ru>
State Research Institute of Aviation Systems,
7, Viktorenko Str, Moscow, 125319, Russia

Abstract. Avionic industry in Russian Federation faces difficulties in organizing the reliable instrumental support of development processes. State-wide active direction on digitalization of the economy doesn't facilitate the issue solving. The choice of software tools is an important component of success while developing complex certifiable software such as aircraft onboard systems. The same situation could be observed in other industries as well. Nowadays the Russian IT-market provides a sufficient amount of different software that can cover the development lifecycle processes of complex certifiable software for avionics in a varying degree. This article analyses the current situation on Russian software market and the impact of import substitution policy of Russian Federation on software developers and consumers – industrial enterprises. Details of regulation document DO-178C for onboard software development are considered to show the importance of correct choice of project's instrumental landscape. Certain types of specialized software tools for development processes automating are considered. Authors identified the basic groups of tool functionality that provide support for the development lifecycle of onboard software. The Russian and foreign PLM (Product Lifecycle Management) and PDM (Product Data Management) systems and other software were examined for compliance with the necessary functionality. For comparative analysis the method based on additive verification of software by criteria was proposed. Research results allowed authors to make a conclusion about current Russian software level in comparison with worldwide analogues. Also some prospects of Russian software further evolution have received justification based on results of this research. Recommendations for the directions of software development and completion are given. The analysis, presented in the article, can be useful for avionic and other industries enterprises which need to choose some software for support the development lifecycle processes in new and ongoing projects of complex systems development. Also specialists who are interested in the current state of Russian IT industry can find some valuable information in this article.

Keywords: software; software analysis; software comparison; Russian software; lifecycle; development lifecycle; onboard systems; onboard software; import substitution; certification; PLM; PDM; complex systems; DO-178C; additive method.

For citation: Gorelits N.K., Gukova A.S., Krasnoshekov D.V. Analysis of Russian software supporting onboard systems development lifecycle in context of import substitution policy. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 175-190. DOI: 10.15514/ISPRAS–2020–32(2)–14

Анализ российского программного обеспечения для поддержки жизненного цикла разработки бортовых систем в условиях политики импортозамещения

Н.К. Горелиц, ORCID: 0000-0003-0974-7146 <nkgorelits@2100.gosniias.ru>
А.С. Гукова, ORCID: 0000-0002-8323-5993 <asgukova@2100.gosniias.ru>
Д.В. Краснощеков, ORCID: 0000-0003-2254-2719 <dvkrasnoshekov @2100.gosniias.ru>
Федеральное государственное унитарное предприятие
«Государственный научно-исследовательский институт авиационных систем»
125319 Россия, Москва, ул. Викторенко, д. 7

Аннотация. Авиационная отрасль в Российской Федерации сталкивается с трудностями организации инструментальной поддержки процессов разработки. Активный курс государства в сторону цифровизации экономики не способствует решению данной проблемы. Выбор инструмента является важным критерием успешной разработки сложного сертифицируемого ПО - бортовых авиационных систем. Подобная ситуация наблюдается и в других отраслях промышленности. В настоящее время на российском ИТ-рынке представлено достаточное количество программного обеспечения, способного в различной степени покрыть те или иные процессы жизненного цикла разработки сложного сертифицируемого программного обеспечения в сфере авионики. В рамках данной статьи анализируется текущая ситуация на российском рынке программного обеспечения и влияние политики импортозамещения Российской Федерации на разработчиков программного обеспечения и его потребителей - промышленных предприятий. Детали нормативного документа КТ-178С для разработки бортового программного обеспечения показывают важность выбора инструментальной среды проекта. Авторы определили основные группы функциональных возможностей инструментов, которые обеспечивают поддержку жизненного цикла разработки бортового программного обеспечения. Российские и зарубежные PLM (Product Lifecycle Management) и PDM (Product Data Management) системы, а также другое программное обеспечение были проверены на соответствие необходимой функциональности. Для сравнительного анализа был предложен метод, основанный на аддитивной верификации программного обеспечения по заданным критериям. Результаты исследований позволили авторам сделать вывод о современном уровне российского программного обеспечения в сравнении с зарубежными аналогами. На основании результатов анализа получили обоснование перспективы дальнейшей эволюции российского программного обеспечения. Также результаты анализа позволили сформулировать рекомендации по направлениям разработки и доработки программного обеспечения и систем. Анализ, представленный в статье, может быть полезен для предприятий авионики и других отраслей, перед которыми стоят задачи выбора программного обеспечения для поддержки процессов жизненного цикла разработки в новых и текущих проектах разработки сложных систем. Также данные о продуктах и выводы могут быть полезны специалистам, интересующимся текущим состоянием российской ИТ-индустрии.

Ключевые слова: программное обеспечение; анализ программного обеспечения; сравнение программного обеспечения; российское программное обеспечение; жизненный цикл; жизненный цикл разработки; бортовые системы; бортовое программное обеспечение; импортозамещение; сертификация; управление жизненным циклом продукта; PLM; управление данными о продукте; PDM; сложные системы; КТ-178С; аддитивный метод.

Для цитирования: Горелиц Н.К., Гукова А.С., Краснощеков Д.В. Анализ российского программного обеспечения для поддержки жизненного цикла разработки бортовых систем в условиях политики импортозамещения. Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 175-190 (на английском языке). DOI: 10.15514/ISPRAS–2020–32(2)–14

1. Introduction

In addition to the high requirements for the reliability and safety of developing software and products, another huge problem in the production of certified products for domestic avionics is the modern governmental policy on technological independence.

Import substitution and digitalization become two main factors in the development of the Russian economy today [1]-[3].

According to the Federal State Statistics Service (Rosstat), if the import substitution policy implemented, the civil aircraft industry expected in 2020 to reduce import dependence from the level of 60-80% to the level of 50-40% [4].

Developers who recently started to automate processes using foreign software are now forced to seek for adaptable software again among Russian products. However, there is no credible information – if there are fully functional qualified analogues of foreign-made software on the Russian market in the form familiar to the user or not. Is the quality of the product or service that provides it sufficient to develop safety-critical products?

Accordingly, the problem of developing certified software, in addition to meeting the requirements and recommendations of numerous aviation documents (international standards, national standards, guidelines, and qualification requirements acting in the industry), has significantly expanded by the need to comply with the course of technological independence in the Russian Federation.

This article provides an actual overview and analysis of Russian software tools for lifecycle management in the development of certified software. As well there is shown the comparison of Russian software with foreign-made analogues traditionally used in the avionic industry in the Russian Federation and abroad (such as Siemens, IBM, etc.).

The article also provides recommendations that may be in demand by enterprises from different industries that need to migrate from foreign to domestic tool platforms, but first of all avionic industry is main interest for this research.

2. Related work and background

2.1 Related work

Software analysis interests a large number of researchers throughout the Russian Federation and affects different field of activity.

There are not many reviews of software for the avionic industry in the scientific sources available to the general public – that is why the appearance of this review was influenced.

Among the scientific works devoted to the topic of comparative analysis of software, we can distinguish the works of Khubaev G.N. [5][6], Shcherbakov S.M. [7], Boikov S.A. [8], Shirobokova S.N. and Serikov O.N. [9], Lisetsky Yu.M. [10], Maslov Yu.G. [11], Zhukov A.G. [12], Krakovskaya T.A co-authored with Tyurnev A.S. [13], Mukhina E.R. [14], Dzyuba E.A. co-authored with Shibanov S.V. and Gerasina A.I. [15], Ozerkova A.V. co-authored with Trubaeva A.L. and Lebedeva M.Yu. [16].

In their work, the authors used three types of methods for software analysis: methods of comparative analysis of software using absorption matrices and graphs [5]-[9], methods of comparison based on binary tables [11]-[13] and simple text reviews [14]-[16]. Separately we would like to highlight the work [10], where author presented the algorithm for selecting the most optimal method for software analysis based on the approach of pairwise comparison of methods.

Our article uses a method of comparative analysis based on additive verification by criteria for selecting software, similar to the one proposed in [11] and adapted for the purposes of this work. The choice of this method is due to the fact that in the case of analyzing tool samples for a small list of parameters, it is quite visual, but at the same time it is not a labor-intensive method to reflect the functional features of the software.

The data obtained as a result of the analysis should first of all be of practical significance for direct users of software (developers, project managers, etc.). In the future, the authors would be interested to develop this topic deeply and to compare software using the method proposed by G.N. Khubaev

[5] and his followers, in order to obtain results for comparing both approaches to analysis and draw conclusions about the applicability of different types of analysis methods.

2.2 Background

Work described in this article was started in GosNIIAS in 2018 in the scope of one of the research and development projects.

First step was made in 2018 and consisted in identification and justification of criteria, which configuration management process puts forward as a requirements for IT-landscape of certifiable product development (especially in avionic industry) [17]. Set of selected criteria was used for analysis of popular requirements management instruments. Results were published and reported during SYRCoSE 2018 conference and some other scientific and practical events.

Next step touched upon the topic of requirements management and its importance for projects in avionic industry [18].

Cursory review of lifecycle management software for certifiable aviation software development was made by authors in 2018 and reported on conference III All-Russian Scientific and Technical Conference «Modeling of aviation systems» [19].

Then special review of requirements management tools for development of safety-critical systems was made by one of the authors and specialists from ISP RAS and published in 2019 [20]. Analytic work was continued and its approach, details and results are described in this article.

3. Research method

In this paper the analysis based on additive comparison was chosen as the research method in order to compare Russian and foreign software. Analysis consisted of the following steps.

1. **Formulation of the problem.** At this step some aspects and reasons of tools choice complexity were formulated.
2. **Overview of software and selection of two sets – Russian software and foreign software.** Software sets were based on the current state of the IT market, available for estimation from open sources data in the Internet, and the authors' knowledge on the current situation at some domestic manufacturing enterprises, which were accumulated as a result of their professional activity. During the software choosing main preferences were given to the software, which is often used in avionic industry enterprises.
3. **Selection of a list of functions for software examination.** Functions for software examination were chosen based on requirements, regulations and recommendation of industry standards and best practices.
4. **Analytic research publicly available from open sources data about software, empirical test of tools which were available for authors.** An analysis of the compliance of the tool with the tested features was performed with:
   - analytical research of information published in open sources available for everyone in the Internet – websites of software developing companies, reference and help materials,
   - interaction with software providers,
   - attraction authors' own practical experience with some of selected software.

   Found results for both sets were added into the table. Table consists of tools in rows and functions in columns. The following notation was chosen in the table:
   - + - function is supported by tool,
   - – - function is not supported by tool,
   - –, I - function is not supported by tool, integration with third-party tools is required.

5. **Counting the formulas for analysis and visualization of data on the diagrams.** For counting and visual analysis "+" value was taken as 1, "–" value was taken as 0, "–, I" value was taken as 0.5. Data from the table was examined in two projections with formulas (1) and (2):

$$\forall j = \overline{1,m}: function_j = \sum_{i=1}^{n} x_{ij} \quad (1)$$

$$\forall i = \overline{1,n}: tool_i = \sum_{j=1}^{m} x_{ij} \quad (2)$$

Where:

- $\boldsymbol{n}$ is a size of tools set (with letter $\boldsymbol{i}$ as index),
- $\boldsymbol{m}$ is a size of functions list (with letter $\boldsymbol{j}$ as index),
- $\boldsymbol{x_{ij}}$ are values from the table cells,
- $\boldsymbol{tool_i}$ is summed value for each tool,
- $\boldsymbol{function_j}$ is summed value for each function.

6. **Fixing conclusions and recommendations.**

4. Comparative analysis of software

4.1 Formulation of the problem

Before considering the tools for managing the software development lifecycle it is necessary to refer to the aviation regulation document DO-178C [21] and its Russian analogue Qualification requirements part 178C [22] as an example of tools choice complexity.

DO-178C defines the rules for the organization of onboard aviation software development processes which are necessary for successful achievement an acceptable level of confidence in product safety and to confirm this level by passing certification and obtaining special aviation certificate. DO-178C defines such important goals like change management and quality management for all of the lifecycle phases.

Regulation document DO-178C specifies the processes of onboard development lifecycle: their definition, input and output data, recommended activities, criteria for the transition between them and many other useful details. But DO-178C does not prescribe the developers of certified software the preferred models of software development lifecycles and the interaction between them. Perhaps this is the reason why the choice of tools becomes such a time-consuming decision for airborne systems software developers, causing numerous disputes, questions and consequences **Ошибка! Источник ссылки не найден.**. In the case of the wrong choice of tool, the cost of resources for changing the IT-landscape and subsequent changes in the project can be very significant especially on the last stages because of the need for certification to start the whole project from almost the very beginning with new set of software. This inevitable decision will cause the loss of pace and the competitive advantage in the market as a result.

In this work we wouldn't analyze tools for compliance with DO-178C requirements. Partly this work was done in other articles of authors [17][18][20].

The next sections of this article describe some packages of Russian software that partially or fully cover the entire software development lifecycle. As well there is maid a comparison of chosen software tools functionality on a set of features with foreign analogues.

For the other parts of aircraft processes such as level of hardware or level of the whole system the developer should analyze lifecycle processes by himself. All of conclusion from this article could be applicable for other levels of processes after some analysis and adaptation if it needs. For

example, lifecycle processes for airborne hardware with regulation document DO-254 [24][25] aren't significantly differ from software lifecycle processes.

Product lifecycle management systems (managing product data) support the full cycle of product and software development and have advanced functionality compared with single-process targeted tools.

Systems, supporting the full cycle of product and software development, include such software groups as Product Data Management systems – PDM systems, Product Lifecycle Management systems – PLM systems, and Collaborating Lifecycle Management systems – CLM systems.

During the analysis two sets of domestic and foreign PLM/PDM systems were formed. Chosen systems cover a similar functional basis of the lifecycle processes. And also a list of functions was selected, which must be automated by tools. The results of the analysis are given below (Table 1).

4.2 Overview of software and sets selection

This subsection contains briefly overview of some CIS-made but in general Russian systems. Complete description could be found on developers sites. These systems will be analyzed below.

4.2.1 T-FLEX DOCs PLM

T-FLEX DOCs software is a scalable solution for product lifecycle management (PLM) and enterprise organization [26]. The solution is based on a set of software T-FLEX CAD/CAM/CAE/CAPP/PDM/CRM/PM/MDM/RM/ – a set of software supplied by one manufacturer – Russian company «Top Systems». It makes possible to organize a single environment for design and technological document flow, design and production preparation.

The solution «T-FLEX DOCs» includes the following capabilities:

Engineering Processes and Design Management; General Office and Desk Workflow; Enterprise Knowledge Management and Archive; Project Management, Cost and Resource Planning; Mail and Tasks, Workflow Management; Integration with ERP Systems; Managing Company Product Range, Corporate Data and Classifiers; Product Structure Management, Bill of Materials, Configurations and Versions; Integration with Major MCAD Systems; Customized Information Systems.

4.2.2 Full Lifecycle Management System: «Digital enterprise» (TIS: Digital enterprise)

It is domestic protected system for managing the full lifecycle of products «Digital enterprise», developed by Russian Federal Nuclear Center – «Rfyats-Vniief» [27].

The software product «Digital enterprise» includes:

Complex of software for digital enterprise resource management, digital enterprise personnel management, production management system, performance management system based on BI-solutions, project and program management, integration platform, regulatory and reference information management, portal services, product lifecycle management, PDM system, protected operating system «Synergy 1.0».

4.2.3 Soyz PLM

Soyuz-PLM is a system for managing engineering and technical information throughout the product lifecycle, developed by Russian company – «Programsoyuz» [28]. Soyuz-PLM is a software package designed to solve various problems of engineering data management in the field of mechanical engineering, instrumentation, architecture, construction and related fields.

The main features are dynamic configuration of the data model as the enterprise develops, the ability to work in a geographically distributed environment, access differentiation and management, and the design of text-based technical documentation for PLM data, integration with external

applications, management of processes and regulated procedures, document management, secure storage of engineering data.

4.2.4 IPS PLM (Intermech Professional Solutions)

A universal information system for product lifecycle management based on enterprise-level INTERMECH solutions designed to manage engineering data and provide information support for a product at various stages of its lifecycle [29]. Currently, IPS PLM is used in various industries – mechanical engineering, instrumentation, industrial and civil construction, nuclear industry, and the military-industrial complex. The software was developed by Belorussian company «Intermech».

4.2.5 Appius-PLM

Integrated information system for product lifecycle management and ERP regulatory framework on the 1C platform: Enterprise 8.3, developed by Russian company «Appius» [30]. Appius-PLM - the solution for managing entire enterprise as a single complex, created based on experience in the development and implementation of CAD/CAM/CAPP/PDM/PLM systems, which allows including design and technological divisions in a single information space of the enterprise, with a single database.

4.3 Selection the PLM/DPM tools for comparison

Among Russian systems, supporting the full cycle of product and software development – PLM/PDM systems we can distinguish the following set: T-FLEX DOCs, Full lifecycle management system «Digital Enterprise» (TIS: Digital Enterprise), APPIUS PLM, Lotsman PLM, Lotsia PLM, Soyuz PLM etc. Also it was decided to include in the analysis the Belorussian software IPS PLM, as the most common system in the CIS countries. The following tools were chosen for the set **TOOLS 1**: T-FLEX DOCs [26], Full lifecycle management system «Digital Enterprise» [27], Soyuz PLM [28], IPS TDM| PDM| PLM | Workflow (Automated Control Systems for Design and Technological Preparation of Production) [29], APPIUS PLM [30].

Among foreign systems with similar functions for the full cycle of product and software development the following systems were chosen for analysis: Siemens Team Center PLM [31], PTC Windchill PLM [32], Dassault Systemes Enovia [33], SolidWorks Enterprise PDM [34]. Also it was decided to include integral solution from IBM - IBM Rational Collaborating Lifecycle Management [35] to the foreign set. Because some components of this solution (such as DOORS / DOORS NG, Change / Synergy, Team Concert, Rhapsody, Test RealTime and others) are most often used at aviation enterprises in the Russian Federation either in separate way of using or in some varies of integration (more rarely). These tools put together a set **TOOLS 2**.

4.4 Selection the list of functions for comparison

Industry regulations [21]-[25][37]-[41], national and international standards [42]-[45], best practices and requirements from enterprises and users for PLM/PDM systems and separate types of tools, accumulated by authors during their work way, were analyzed and gave a huge list of useful and necessary functions for tools for the full lifecycle coverage. Some functions were chosen to make a comparative analysis of systems in sets **TOOLS 1** and **TOOLS 2** – necessary features for automation or support the full cycle of product and software development. These features made up a list of parameters **PLM/PDM PARAMS** (short name of parameter for diagram put in braces):

- ability to integrate with CAD systems (short: CAD),
- reference data management (short: RDM),
- ability to make custom agreement processes with electronic signature and other types of workflows (short: Workflow),

- technological support of production (short: Prod.support),
- requirements management (short: RM),
- quality management (short: QM).

4.5 Comparative analysis

This chapter presents comparative analysis of selected tools from the sets **TOOLS 1** and **TOOLS 2** with selected criteria from the list of functions PLM/PDM PARAMS. Table 1 contains results of experiments and estimation of software, which will be used below for counting with formulas (1) and (2) and further visualization.

Table 1. Comparative analysis of PLM/PDM systems: functions

Name of software	Russian software	CAD	RDM	Workflow	Prod. support	RM	QM
Set TOOLS 1: Russian/CIS software							
T-FLEX DOCs	+	+	+	+	+	+	–
Full lifecycle management system "Digital Enterprise"	+	+	+	+	+	+	–
Soyuz PLM	+	+	+	+	+	+	–
IPS TDM| PDM| PLM | Workflow	+	+	+	+	+	–	–
Appius PLM	+	+	+	+	+	–	–
Set TOOLS 2: foreign software							
SolidWorks Enterprise PDM	–	+	+	–, I	+	–	–
IBM Rational Collaborating Lifecycle Management	–	–	–	+	–	+	+
Dassault Systemes Enovia	–	+	–, I	–, I	+	+	–
PTC Windchill PLM	–	+	–, I	+	+	+	+
Siemens Team Center PLM	–	+	+	–, I	+	+	+

4.6 Counting and visualization

Then in first case tools score from each set was summed up for each parameter from the set of functions. Here the formula (1) was used.

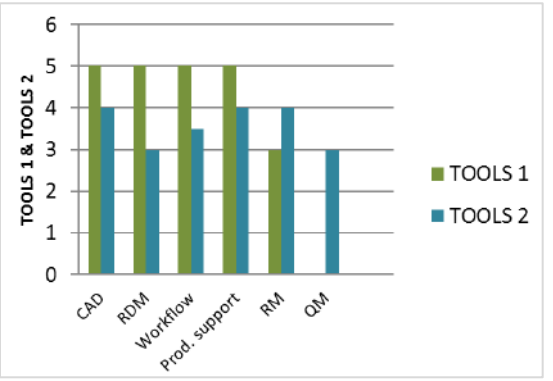

Fig. 1. How many tools from sets TOOLS 1 and TOOLS 2 have functions from PLM/PDM PARAMS list

On the Fig. 1 there is a comparison between two sets **TOOLS 1** and **TOOLS 2** by **PLM/PDM PARAMS** – how many tools from each set have selected functions. Fig. 1 shows that there are some more attractive features for product developers, whereas other features are very important as well.

In the second case values of features for each software were summed. Both sets **TOOLS 1** and **TOOLS 2** were compared between each other and with some hypothetical Reference tool – how many functions from PLM/PDM PARAMS list has each software from sets **TOOLS 1** and **TOOLS 2**. Reference tool has all features - 6. Here the formula (2) was used. Fig. 2 shows that no one tool from both sets has all features. And also it is seen on both Fig. 1 and Fig. 2 that in set TOOLS 1 (Russian software) more instruments have necessary functions than in set **TOOLS 2** (foreign software).

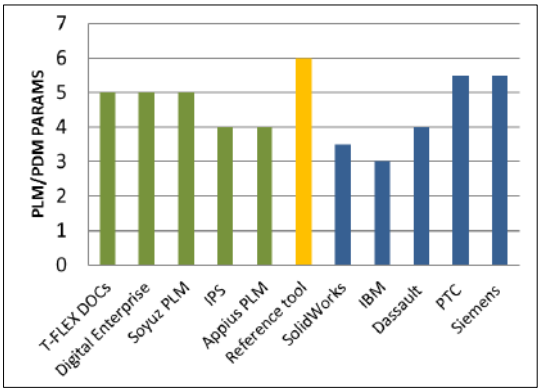

Fig. 2. How many functions software from sets TOOLS 1 and TOOLS 2 covers

4.7 Intermediate conclusion

All of the listed software products have much in common by functionality. Foreign software, unlike Russian, does not take into account the existing mentality and work culture of specialists in the post-Soviet space and enterprises. Domestic software was created to solve problems and based on the needs of only Russian enterprises, considering their specificity and the established regulatory framework.

Because of this fact the functionality and technological processes in domestic PLM and PDM systems are developed with better attention to local needs and rules. The built-in processes, document flow, data formats, reports and etc. more closely correspond to Russian standards.

In order to support processes, which are common to all developers, and also domestic-specific Russian processes, software from foreign vendors must either be integrated with Russian modules or develop add-ons with similar functionality. All of these additional activities will lead to extra costs and time losses during implementation of software that is already very expensive.

Nowadays Russian aviation enterprises solve the problems of supporting the full lifecycle of their products. And these problems go beyond the design and technological preparation of production. That is why Russian PLM systems are also growing dynamically. However, they are moving after Russian enterprises - their development takes place and has a direction depending on the needs of specific customers. The solutions of foreign vendors such as DASSAULT Systemes, Siemens PLM, PTC, IBM and other foreign developers contain the experience of Western industrial enterprises, which are much more advanced than domestic ones due to the earlier and balanced implementation of approaches and practices of systems engineering [42]**Ошибка! Источник ссылки не найден.**[47].

Nevertheless, the teams of domestic developers deeply understand the need to automate the processes of lifecycle and the provision of end-to-end technologies in their software. Today

developers already add to their product lines such modules as, for example, requirements management (T-FLEX DOCs, Soyuz PLM) or complaint management (T-FLEX DOCs). Also it makes sense to highlight the software product 8D from ASCON [48]. 8D wasn't included to the set **TOOLS 1** but nevertheless 8D is one of the few products in the Russian Federation in which quality management support has appeared.

All these facts allow us to surely conclude that domestic PLM and PDM systems are ready to correspond foreign analogues, and even bypass them according to a number of criteria. Moreover, Russian software is closer to Russian production realities, which probably makes the introduction of such software less "traumatic" for users.

5. Recommendations for choosing software

In the context of the import substitution policy in Russian Federation, it is necessary to consider additional parameters besides to functionality when choosing software. The same additive method as proposed in this article was used for analyzing and visualizing the result.

Some characteristics were chosen to estimate the readiness of Russian software to replace foreign analogues from the point of view of the import substitution program for increasing technological independence of Russian Federation. These characteristics made up a list of parameters **SUBSTITUTION PARAMS**:

- registration in the Russian Register (Unified Register of Russian programs for electronic computers and databases) [49];
- earning the certificate of FSTEC of Russia (Federal Service for Technical and Export Control) [50];
- integration with other software tools or built-in software functions (CAD/ECAD systems, workflow systems, configuration management systems, master data management systems, ERP systems, etc.)
- Russian-speaking technical support;
- compliance with the statement of the problem – providing functions which cover necessary aspects of lifecycle processes and features of foreign analogues;
- successful implementation to the aviation industry enterprises;
- successful implementation to the other industries enterprises.

Just to show the example of these recommendations applying here will be given one more analysis – software tools from the set **TOOLS 1** were taken for analysis with **SUBSTITUTION PARAMS**. The results of the analysis are given below (see Table 2, Fig. 3 and Fig. 4).

Table 2. Comparative analysis of Russian software: import substitution

Name of software	Russian Register	Certificate of FSTEC	Integration	Russian support	Aviation	Other industries
T-FLEX Docs	+	–	+	+	+	+
Full lifecycle management system "Digital Enterprise"	+	+	+	+	–	+
Soyuz PLM	+	–	+	+	–	+
IPS TDM| PDM| PLM | Workflow	–	+	+	+	+	+
Appius PLM	+	–	+	+	+	+

Similarly with the previous analysis Fig. 3 and Fig. 4 were formed. Fig. 3 shows how many tools from set **TOOLS 1** automate each feature from **SUBSTITUTION PARAMS** list. It is seen that not

all instruments satisfied all parameters - some parameters are more difficult to satisfy than others. Here the formula (1) was used.

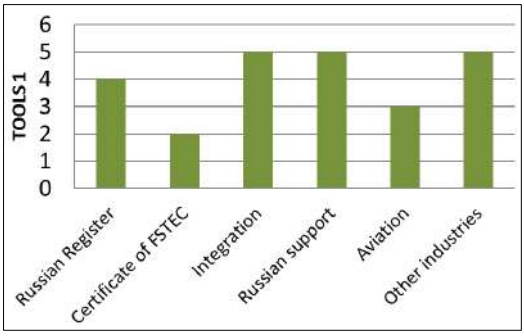

Fig. 3. How many tools from set TOOLS 1 have features from SUBSTITUTION PARAMS list

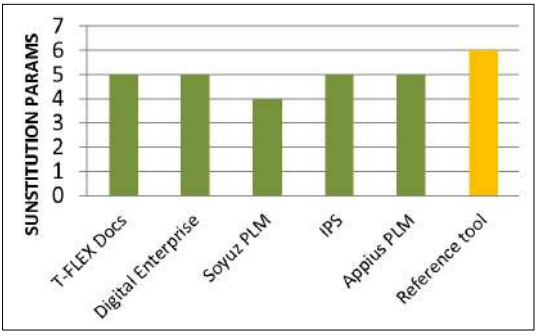

Fig. 4. How many features software from set TOOLS 1 covers

Fig. 4 shows how many features from **SUBSTITUTION PARAMS** list has each software from set **TOOLS 1**. Instruments were compared between each other and with some hypothetical Reference tool. It is seen that there is no real instrument in set **TOOLS 1**, which could satisfy all import substitution parameters like Reference tool with 6 score. Here the formula (2) was used.

Some more facts should be carefully checked out and used as potential parameters for comparison during choosing software: comparison with analogues, geographical location, staff for implementation and history of successful implementations. Here is a short description of these facts below as text just not to complicate the table and figures.

It is important to know during choosing a software if any comparisons with foreign analogues were carried out. For example, what percent of the functions of a foreign analogue are covered by software under review? And what plans for further development are existed — whether it is planned by developer to cover 100% of the functions in the near future?

For nowadays situation with import substitution the aspect of geographical location is very important as well. Enterprises should pay attention on it – if developer's servers and software development itself are located on the territory of the Russian Federation. For example, if the software was developed by a foreign company, and then a company in the Russian Federation bought it – presently software's origin may become an obstruction to its implementation in Russian governmental enterprises in the boundaries of import substitution.

Additional advantages to software developer while choosing software are a pool of partners or their own experienced staff for implementation and training and a history of successful implementation in the avionic industry or related industries.

6. Conclusion

In Russian Federation not only avionic industry but also aerospace, nuclear, railway, automobile, shipbuilding, medical and other industries have difficulties in automating development processes, despite the active digitalization course of the state.

Choosing the toolset that will be used throughout the project is an important element in planning the development of complex certified software. IT-landscape has a significant impact on the success and cost of the entire project.

Software developers in the aviation sector do not have a common opinion about what should be an ideal tool for managing the processes of software and product development and its entire lifecycle. Some developers want to see a single platform to support development lifecycle and complete automation of all processes. Another part of developers insists that more processes should be controlled manually, because not all processes can be properly automated at the moment for various reasons.

There is no consensus on this issue - regulatory documents usually do not impose the development process or preferred lifecycle models by the developer, but only provide recommendations for the organization of this process and its limitations.

At the time of writing this article, there is no ideal tool for solving problems of automation the development processes in the industry, which has proven itself in practice in the Russian Federation - the concept of unity of digital platforms is only at the beginning of its path.

But it is already being discussed at the state level, so we have chance to see its successful implementation in the future.

As in the case of foreign software – it is not enough just to buy and install "boxed" Russian software– it also requires huge labor costs for adapting processes, methodology, implementation and training of enterprise personnel.

There is bit public information about the successful implementation of Russian systems in aviation. There is only information published on the official websites of vendors that indicates the fact of purchasing software. The last decade authors' experience of interaction with Russian industrial enterprises and software suppliers has shown that the purchase of software licenses does not guarantee the use and successful implementation on the enterprise. Despite it at the same time the project can be listed on the developer's website as an example of successful implementation because of business interests.

However today there is a sufficient amount of software available on the Russian market that can cover part of the software lifecycle processes in the development of complex certified software. Many software developers are open to interaction with enterprises and users and, reviewing the trend of recent years, are ready to refine their software in accordance with industry requests. For example, important modules for requirements management and quality management have already appeared in some Russian systems although previously these modules were missing.

Based on the results of the analysis, we can make a conclusion about the active development and use of domestic Russian systems and tools with an insufficient level of integration between them. The effectiveness of using an isolated tool decreases due to the need to convert, and sometimes re-enter existing (in another system) source data. The same output can be noted for using the results of work that require additional actions to transfer data to the system for further use.

Russian software covers the automation of technological processes of domestic Russian enterprises better than foreign software. And today Russian software is not inferior to foreign manufacturers in integration with design data. And it is pleased to note that some Russian analogues of CAD and ECAD systems have been developed already.

The results of the analysis and suggested recommendations given in this article should help enterprises and organizations from various industries at the initial stages of choosing import-substituting tools for automating enterprise development processes.

References / Список литературы

[1] Decree of the Government of the Russian Federation of April 15, 2014 N 328 "On approval of the state program of the Russian Federation" Development of industry and increasing its competitiveness", available at: http://ivo.garant.ru/#/document/70643464/paragraph/1:0 (in Russian) / Постановление Правительства РФ от 15 апреля 2014 г. N 328 "Об утверждении государственной программы Российской Федерации "Развитие промышленности и повышение ее конкурентоспособности".

[2] Decree of the Government of the Russian Federation of March 2, 2019 N 234 "On the system for managing the implementation of the national program" Digital Economy of the Russian Federation" http://ivo.garant.ru/#/document/72190034/paragraph/1:0 (in Russian) / Постановление Правительства РФ от 2 марта 2019 г. N 234 "О системе управления реализацией национальной программы "Цифровая экономика Российской Федерации".

[3] Shelomanova P.A., Kuzmin R.A. The state program of import substitution in the Russian economy until 2020. Development and current issues of modern science, Magnitogorsk, no. 5(5), 2017, pp. 72-76 (in Russian) / Шеломанова П.А., Кузьмин Р.А. Государственная программа импортозамещения в российской экономике до 2020 года. Развитие и актуальные вопросы современной науки, no. 5(5), 2017 г., стр. 72-76.

[4] Federal State Statistic Service (Rosstat), URL: https://www.gks.ru / Федеральная служба государственной статистики.

[5] Khubaev G.N. Comparison of complex software systems by the criterion of functional completeness. Programmnye produkty i sistemy (Software & systems), no. 2, 1998, pp. 6-9 (in Russian) / Хубаев Г.Н. Сравнение сложных программных систем по критерию функциональной полноты. Программные продукты и системы, no. 2, 1998 г., стр. 6-9.

[6] Khubaev G. N. Comparison of software products by the criterion of «Performance». Programmnye produkty i sistemy (Software & systems), no. 4, 2008, pp. 27-33 (in Russian) / Хубаев Г.Н. Сравнение программных продуктов по критерию «Производительность». Программные продукты и системы, no. 4, 2008 г., стр. 27-33.

[7] Shcherbakov S.M. The method of analysis of complex systems by the criterion of functional completeness: expansion and adaptation. System Management. 2010. №2(8), pp. 1-20 (in Russian) / Щербаков С.М. Метод анализа сложных систем по критерию функциональной полноты: расширение и адаптация. Системное управление, вып. 2(8), 2010 г., стр. 1-29.

[8] Boykov S.A. Models and valuation methods of the functional completeness of information systems for the state institutions in the social sphere. Business. Education. Law. Bulletin of Volgograd Business Institute, no. 4, 2014, pp. 231-235 (in Russian) / Бойков С.А. Модели и методы оценки функциональной полноты информационных систем для государственных учреждений в социальной сфере. Бизнес. Образование. Право. Вестник Волгоградского института бизнеса, no. 4, 2014 г., стр. 231-235.

[9] Shirobokova S.N., Serikov O.N. Formal analysis of functional completeness of a system of video analytics. Engineering Journal of Don, no. 1, 2019, pp.33-47 (in Russian) / Широбокова С.Н., Сериков О.Н. Формализованный анализ функциональной полноты систем видеоаналитики. Инженерный вестник Дона, no. 1, 2019 г., стр. 33-47.

[10] Lisetsky Yu.M. Algorithm for comparing the methods of complex quantitative assessment of the quality of complex systems. Programmnye produkty i sistemy (Software & systems), no. 4, 2012, pp.153-156 (in Russian) / Лисецкий Ю.М. Алгоритм сравнения методов комплексной количественной оценки качества сложных систем. Программные продукты и системы, no. 4, 2012 г., стр. 153-156.

[11] Maslov Yu.G. About the methodology for software products comparison. Information Security, no. 2, 2007. pp.56-57 (in Russian) / Маслов Ю.Г. О методике сравнения программных продуктов. Информационная безопасность, no. 2, 2007 г.. стр. 56-57

[12] Zhukov A.G. Comparison of software products based on the analytic hierarchy. European researcher, no. 6, 2011, pp. 934-935 (in Russian) / Сравнение программных продуктов на основе аналитической иерархии. European researcher, no. 6, 2011, стр. 934-935

[13] Krakovskaya T.A., Tyurnev A.S. Comparative analysis of software products for marketing research. Modern technologies, System analysis, Modeling, no. 1, 2007, pp.120-126 (in Russian) / Краковская Т.А., Тюрнев А.С. Сравнительный анализ программныхродуктов для маркетинговых исследовани. Современные технологии, системный анализ, моделирование, no. 1, 2007 г., стр. 120-126

[14] Mukhina E.R. Comparative characteristics of software products allowing management accounting. Actual problems of humanitarian and natural sciences, no. 9, 2014, pp.160-163 (in Russian) / Мухина Е.Р. Сравнительная характеристика программных продуктов, позволяющих вести управленческий учет. Актуальные проблемы гуманитарных и естественных наук, no. 9, 2014 г., стр. 160-163

[15] Dzyuba E.A., Shibanov S.V., Gerasina A.I. Comparative analysis of modern instruments supporting the life cycle of information systems. In Proc. of the International Symposium «Reliability and quality», 2012, pp. 420-426 (in Russian) / Труды Международного симпозиума «Надежность и качество», 2012 г., стр. 420-426.

[16] Ozerkova A.V., Trubaeva A.L., Lebedeva M.Yu. Comparison of software products that can be used in the organization MUP KH "CHISTIC". New science: from idea to result, no. 4-1, 2016, pp. 65-68 (in Russian) / Озеркова А.В., Трубаева А.Л., Лебедева М.Ю. Сравнение программных продуктов, которые могут быть использованы в организации МУП КХ "ЧИСТИК". Новая наука: от идеи к результату, no, 4-1, 2016 г., стр. 65-68.

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

[18] Krasnoshekov D.V., Gorelits N.K., Peskov E.V. Requirements management for software development in the aviation industry. IT-Standard, no. 2, 2018, pp. 12-17 (in Russian) / Краснощеков Д.В., Горелиц Н.К., Песков Е.В. Аспекты управлениятребованиями при разработке программного обеспечения в авиационной отрасли. ИТ-Стандарт, no. 2, 2018 г., стр. 12-17

[19] Gorelits N.K., Gukova A.S. Overview of lifecycle management software for certifiable aviation software development. Abstracts of the III All-Russian Scientific and Technical Conference «Modeling of aviation systems», 2018, p. 223 (in Russian) / Горелиц Н.К., Гукова А.С. Обзор и сравнительный анализ инструментальных средств для управления жизненным циклом при разработке сертифицируемого ПО. Тезисы докладов III Всероссийской научно-технической конференции «Моделирование авиационных систем», 2018 г., стр. 223.

[20] Gorelits N.K., Kildishev D.S., Khoroshilov A.V. Requirement management for safety-critical systems. Overview of solutions. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 1, 2019. pp. 25-48 (in Russian). DOI: 10.15514/ISPRAS-2019-31(1)-2 / Горелиц Н.К., Кильдишев Д.С, Хорошилов А.В. Управление требованиями к ответственным системам. Обзор решений. Труды ИСП РАН, том 31, вып. 1, 2019 г., стр. 25-48.

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

[22] Qualification requirements part 178C. Software requirements for onboard equipment and systems for certification of aviation equipment. M., IAC, 2016, 131 p. (in Russian) / Квалификационные требования КТ-178С. Требования к программному обеспечению бортовой аппаратуры и систем при сертификации авиационной техники. М., АР МАК, 2016, 131 стр.

[23] Solodelov Yu.A., Gorelits N.K. Certifiable onboard real-time operation system JetOS for Russian aircrafts design. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 3, 2017, pp. 171-178 (in Russian). DOI: 10.15514/ISPRAS-2017-29(3)-10 / Солоделов Ю.А., Горелиц Н.К. Сертифицируемая бортовая операционная система реального времени JetOS для российских проектов воздушных судов. Труды ИСП РАН, том 29, вып. 3, 2017 г., стр. 171-178.

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

[25] Qualification Requirements part 254. Guidance on the warranty design of onboard electronics. M., IAC, 2011 (in Russian) / Квалификационные требования КТ-254. Руководство по гарантии конструирования бортовой электронной аппаратуры. М., АР МАК, 2011, 89 стр.

[26] T-FLEX DOCs PLM. Availavle at: https://www.tflex.com, accessed 10.05.2020.

[27] Full lifecycle management system «Digital Enterprise». Availabe at: http://vniief.ru/researchdirections/tisjaok/, accessed 10.05.2020 (in Russian) / СУ ПЖЦ «Цифровое предприятие»).

[28] Soyuz PLM. Availabe at: http://www.programsoyuz.ru/products/system-soyuz-plm/, accessed 10.05.2020 (in Russian) / Союз-PLM.

[29] IPS TDM| PDM| PLM | Workflow. Availabe at: https://intermech.ru, accessed 10.05.2020 (in Russian).

[30] APPIUS PLM. Availabe at: http://www.appius.ru, accessed 10.05.2020 (in Russian).

[31] Siemens Team Center PLM. Availabe at: https://new.siemens.com, accessed 10.05.2020.

[32] PTC Windchill PLM. Availabe at: https://www.ptc.com/en, accessed 10.05.2020.

[33] Dassault Systemes Enovia. Availabe at: https://www.3ds.com, accessed 10.05.2020.

[34] SolidWorks Enterprise PDM. Availabe at: https://www.solidworks.com, accessed 10.05.2020.

[35] IBM Rational Collaborating Lifecycle Management. Availabe at: https://jazz.net, , accessed 10.05.2020.

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

[37] Guideline R-4754A on the development of civil aircraft and systems. M., IAC, 2016, 131 p. (in Russian) / Руководство Р-4754А по разработке воздушных судов гражданской авиации и систем. М., АР МАК, 2016, 131 стр.

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

[39] Guidelines R-4761 for Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, IAC, 2010, 242 pp. (in Russian) / Руководство Р-4761 по методам оценки безопасности систем и бортового оборудования самолетов гражданской авиации. М., АР МАК, 2010, 262 стр.

[40] Integrated Modular Avionics Development Guidance and Certification Considerations (RTCA/DO-297), 2005

[41] Guidelines R-297 for integrated modular avionics development and qualification. M., AP MAK, 2015, 123 p. (in Russian) / Руководство по вопросам разработки и квалификации интегрированной модульной авионики Р-297. М., АР МАК, 2015, 123 стр.

[42] ISO/IEC/IEEE 15288 System engineering — System life cycle, 2015.

[43] ISO 10007 Quality management — Guidelines for configuration management, 2017.

[44] ISO/IEC 12207 Systems and software engineering – Software life cycle processes, 2008.

[45] ISO/IEC/IEEE 29148 Systems and software engineering – Life cycle processes – Requirements engineering, 2011.

[46] Koverninskiy I.V., Kan A.V., Volkov V.B., Popov Yu.S., Gorelits N.K. Practical experience of software and system engineering approaches in requirements management for software development in aviation industry. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 2, 2016, pp. 173-180. DOI: 10.15514/ISPRAS-2016-28(2)-11

[47] Gorelits N.K., Peskov E.V. Analysis of system engineering approaches for complex systems development in aviation industry. Abstracts of the III All-Russian Scientific and Technical Conference «Modeling of aviation systems», 2018, p. 231 (in Russian) / Горелиц Н.К., Песков Е.В. Анализ подходов системной инженерии при разработке сложных систем в авиационной отрасли. Тезисы докладов III Всероссийской научно-технической конференции «Моделирование авиационных систем», 2018 г., стр. 231.

[48] Ascon 8D. Quality management. Available at: https://ascon.ru/products/1248/review/, accessed 10.05.2020 (in Russian) / 8D. Управление качеством – Аскон.

[49] Unified Register of Russian programs for electronic computers and databases. URL: https://reestr.minsvyaz.ru (in Russian) / Единый реестр российских программ для электронных вычислительных машин и баз данных.

[50] Federal Service for Technical and Export Control. URL: https://fstec.ru (in Russian) / Федеральная служба по техническому и экспортному контролю.

Information about authors / Информация об авторах

Natalia Kirillovna GORELITS – engineer of 1 category in department of Advanced systems and avionics integration, received her specialist's degree in applied math and computer science in Lomonosov Moscow State University in 2010. Main research interests: system engineering, requirements management, implementation of methodologies, processes and tools, support of the complex systems lifecycle management and certification processes.

Наталья Кирилловна ГОРЕЛИЦ – инженер 1 категории подразделения Перспективных разработок и комплексирования авионики. Она защитила диплом специалиста в области прикладной математики и информатики в Московском государственном университете им. М.В. Ломоносова в 2010 году. Область научных интересов включает системную инженерию, управление требованиями, внедрение методологий, процессов и инструментов, поддержку процессов жизненного цикла разработки и сертификации сложных систем.

Aleksandra Sergeevna GUKOVA – engineer in department of Advanced systems and avionics integration since 2018, works on the software implementation at the enterprises of the Russian

Federation since 2010. Her research interests include system engineering, software for lifecycle support, certification of civil aircraft and avionics equipment, digitalization.

Александра Сергеевна ГУКОВА – инженер подразделения Перспективных разработок и комплексирования авионики с 2018 года, занимается внедрением программного обеспечения на предприятиях РФ с 2010 года. Сфера научных интересов включает системную инженерию, внедрение программного обеспечения для поддержки жизненного цикла, сертификацию гражданских воздушных судов и бортового авиационного оборудования, цифровизацию.

Dmitry Vladimirovich KRASNOSCHEKOV – leading engineer in department of Advanced systems and avionics integration. His activities and research interests are related to the implementation of development processes under certification requirements in the production of avionics.

Дмитрий Владимирович КРАСНОЩЕКОВ – ведущий инженер подразделения Перспективных разработок и комплексирования авионики. Его деятельность и научные интересы связаны с постановкой процессов разработки в рамках требований сертификации при производстве авионики.