ТРУДЫ

ИНСТИТУТА СИСТЕМНОГО ПРОГРАММИРОВАНИЯ РАН

PROCEEDINGS OF THE INSTITUTE FOR SYSTEM PROGRAMMING OF THE RAS

ISSN Print 2079-8156
Том 33 Выпуск 6

ISSN Online 2220-6426
Volume 33 Issue 6

Институт системного
программирования
им. В.П. Иванникова РАН

Москва, 2021

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:

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/

С о д е р ж а н и е

Table of Contents

DOI: 10.15514/ISPRAS-2021-33(6)-1

Capabilities and Restrictions of Software Model Checkers

E.M. Novikov, ORCID: 0000-0003-3586-3140 <novikov@ispras.ru>
Ivannikov Institute for System Programming of the RAS,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. Software model checkers enable automatic detection of violations of specified requirements in programs as well as formal proof of correctness under certain assumptions. These tools actively evolve two last decades. They were already successfully applied to a bunch of industrial projects, first of all to kernels and drivers of various operating systems. This paper considers an interface of software model checkers, their unique capabilities as well as restrictions that prevent their large-scale usage on practice.

Keywords: software model checking; formal verification; requirements specification; violation witness.

For citation: Novikov E.M. Capabilities and Restrictions of Software Model Checkers. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 7-14. DOI: 10.15514/ISPRAS–2021–33(6)–1

Возможности и ограничения инструментов верификации моделей программ

Е.М. Новиков, ORCID: 0000-0003-3586-3140 <novikov@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Инструменты верификации моделей программ позволяют автоматически искать нарушения специфицированных требований в программах, а также доказывать их корректность формально при выполнении определенных условий. Данные инструменты развивались достаточно активно два последних десятилетия. За это время они были успешно использованы в ходе верификации нескольких промышленных проектов, в первую очередь ядра и драйверов различных операционных систем. Данная статья рассматривает интерфейс инструментов верификации моделей программ, их уникальные возможности, а также ограничения, которые затрудняют их широкомасштабное практическое применение.

Ключевые слова: верификация моделей программ; формальная верификация, спецификация требований; свидетельство нарушения

Для цитирования: Новиков Е.М., Возможности и ограничения инструментов верификации моделей программ. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 7-14. DOI: 10.15514/ISPRAS–2021–33(6)–1

1. Introduction

Software model checking helps to find violations of specified requirements in programs non-interactively and prove program correctness formally under certain assumptions. These capabilities are highly demanded by the industry since conventional quality assurance approaches either fail to reveal all faults in programs under verification [1] or their usage needs enormous efforts [2, 3].

Developers of software model checkers (they will be also called *verification tools* below) have been forming an active community since the appearance of first such tools at the beginning of the century. One of the most important steps in this direction was the organization of a series of annual competitions on software verification (SV-COMP). The first competition in 2012 attracted about a dozen of developer teams from leading universities and research centers from all over the world [4]. Since then, the number of participants has been growing steadily and already 27 teams participated in SV-COMP 2021 [5].

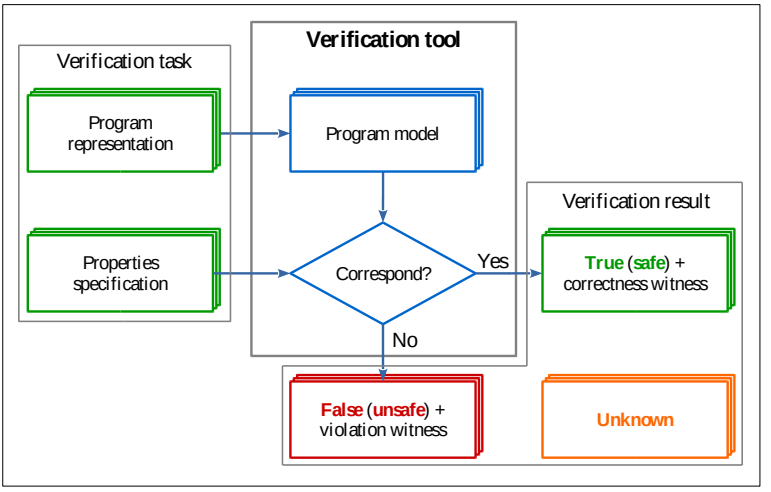


Fig. 1. Verification tool interface and basic workflow

Fig. 1 illustrates an interface and a basic workflow of a verification tool. The interface is specified in detail in SV-COMP rules [6]. Though, it changes from year to year but highly likely most major decisions have been done already. The verification tool gets as an input a so-called *verification task* that consists of a program representation, e.g. files with program sources or LLVM bitcode, and a properties specification. The former is based on the source code of the target program and the latter allows to specify requirements to check. The verification tool builds a program model on the base of the program representation and checks it against the properties specification. This is performed fully automatically.

As a result the verification tool provides a *verdict* that answers the following question: «Does the program satisfy the specification?» In case of successful verification, it also outputs a *witness* in addition to the verdict. Witnesses are machine-readable files containing parts of formal evidences that can be validated automatically or studied manually to confirm or to reject verification results. If the verification tool cannot provide a definite answer, say, because it depletes allotted computational resources such as CPU time or memory, then it gets terminated and the verdict is set to *unknown*.

Following sections consider particular aspects of the verification tool interface, capabilities and restrictions of verification tools as well as extra features required for application of them to industrial programs.

2. Supported Programs

Verification tools support verification of software developed in various programming languages. This paper considers verification of C programs exclusively since they are quite widespread among industrial programs requiring a very high level of quality.

According to SV-COMP rules a verification task should contain a single C source file. This file should be prepared in advance so that verification tools can take it as input without any additional

processing. Since industrial programs usually contain many source files, users should preprocess and merge them beforehand. Most programs interact with their environment including users, libraries, other programs, hardware and so on. This interaction should be incorporated into the C source file of the verification task since verification tools operate non-interactively, they do not communicate to program environments and they dislike undefined behavior. SV-COMP rules do not consider how this can be achieved. This topic will be considered further in Section 6.

Programs developed in GNU C constitute the lion share of the SV-COMP benchmark suite with the combined size of approximately 100 MLOC of preprocessed code. Thus, many verification tools that participate in SV-COMP have a high-quality support for GNU C programs, though some specific extensions, e.g. attributes, can be unsupported. Support for other compiler extensions depends on used front-ends primarily.

On average verification tasks are 3 KLOC in size. That's why users can hardly expect that they will be able to use verification tools out of the box for industrial programs that contain hundreds and thousands of KLOC. The SV-COMP community does not provide users with ideas and auxiliary tools to tackle the given issue. In following sections performance and scalability of verification tools will be considered in more details.

Many industrial C programs use various parallel programming means. This paper treats just multithreading. Accurate verification of multithreaded programs is a much harder task in comparison with verification of sequential programs. Later a special attention will be paid to the given issue.

3. Supported Requirements

This paper focuses on verification of programs against non-functional requirements which violations can result in critical failures like denial of service, privilege escalation and data breaches[1]. For instance, it is vital to detect such common weaknesses of C programs as buffer overflows and null pointer dereferences. Other examples of non-functional requirements are rules of correct usage of an API, which are also often violated and which violations can be rather harmful [7].

Table 1. Formulas describing properties supported by verification tools

Property	**Formula**
Unreachability	*CHECK(init(main()), LTL(G ! call(reach_error())))*
Memory Safety	*CHECK(init(main()), LTL(G valid-free))* *CHECK(init(main()), LTL(G valid-deref))* *CHECK(init(main()), LTL(G valid-memtrack))* *CHECK(init(main()), LTL(G valid-memcleanup))*
Overflow	*CHECK(init(main()), LTL(G ! overflow))*
Termination	*CHECK(init(main()), LTL(F end))*

Verification tools can check programs against safety and liveness properties. SV-COMP defines a common format for specifications of such properties in the form of linear temporal logic (LTL) formulas. Table 1 presents currently supported properties and corresponding formulas. Here *init(main())* represents a program entry point assuming calling function *main()* without parameters. LTL operator *G f* means that formula *f* holds in every state of the program, so, for example, *G ! overflow* means that integer overflow should never happen, and *G ! call(reach_error ())* means that the error function should not be ever called (otherwise, there may be faults in checked programs). If unclear, semantics of other properties and formulas can be clarified using SV-COMP rules.

[1] For thorough checking of program functionality other methods, such as deductive verification, suit better. Similarly, for detecting non-critical non-functional requirements, e.g. coding style violations, there are other good methods and tools.

The SV-COMP community does not suggest any widely accepted means for checking those requirements that do not correspond explicitly to one of the supported properties. Users can either leverage specific capabilities provided by some verification tools, e.g. for finding data races [8], or weave an additional source code into a program either manually or automatically to express requirements using one of the supported properties. For instance, rules of correct usage of a particular API can be formulated as unreachability of the error function like in Fig. 2 and Fig. 3.

```
/* Device driver can register just
   one device at a time. */
int register_device(void) {
    ...;
}

/* Device driver can unregister device
   just after it registers it. */
void unregister_device(void) {
    ...;
}
```

Fig. 2. Original program

```
bool is_device_registered = false;

int register_device(void) {
    if (is_device_registered)
        reach_error();
    is_devi)ice_registered = true;
    ...;
}

void unregister_device(void) {
    if (!is_device_registered)
        reach_error();
    is_device_registered = false;
    ...;
}
```

Fig. 3. Woven in program

If one expresses weakly related requirements using the same property, it is possible to check them simultaneously, but this is not recommended due to the following issues. The first reason for this is that verification tools may build and check substantially different models. It is not an easy task how to distribute available computational resources between these models when they are complex enough. The second reason is that most verification tools stop after they find a first violation of a checked property. So, detecting a first fault or a false alarm can prevent finding other faults. Overall, it may be better to check different requirements independently.

Verification tools can hardly check large and complicated parallel programs. Hopefully, for checking most of requirements it is not necessary to consider all possible interleavings of threads[2]. There are different approaches how to serialize parallel programs. Section 6 presents some related ideas.

4. Verification Accuracy

Verification tools tend to construct program models in a sound way that keeps all errors existing in the source code under verification. However, as a rule they make some assumptions either implicitly or explicitly according to specified configuration options to significantly raise their efficiency. For

[2] Here it is assumed that the target program does not have any concurrency issues.

example, many verification tools can treat undefined functions, e.g. library functions, as functions without side effects returning any value corresponding to their return types. Often this does not affect verification results[3], but sometimes this is not the case, e.g. when undefined functions allocate and initialize memory referred later. Most verification tools do not support the inline assembler. Verification tools implementing bounded model checking unroll loops just to a given number of iterations.

All these assumptions can result in both missing faults and false alarms. Fortunately, verification tools can report possible verification inaccuracies while users can change corresponding configuration options and provide models to bypass these issues at least to some extent.

5. Performance and Scalability

Comprehensive verification is an extremely complicated problem. Usually, it is difficult to predict computational resources necessary for it since an outcome depends on many factors such as code complexity, requirements being checked, verification algorithms, solvers and so on.

SV-COMP rules specify the following limits for each verification task: 15 minutes of CPU time and 15 GB of RAM. Most successful tools can cope with programs of several dozens of KLOC in size within these limits but not always. Significant increase of complexity of source code parts relevant to checked requirements almost always results in an enormous growth of required computational resources and inability to proceed with the same verification scope and precision. This is demonstrated by three quantile plots in Fig. 4. These plots show how many verification tasks can be solved using an appropriate amount of CPU time. One can see that verification tools operate differently but none of them can solve all verification tasks within the specified time limit.

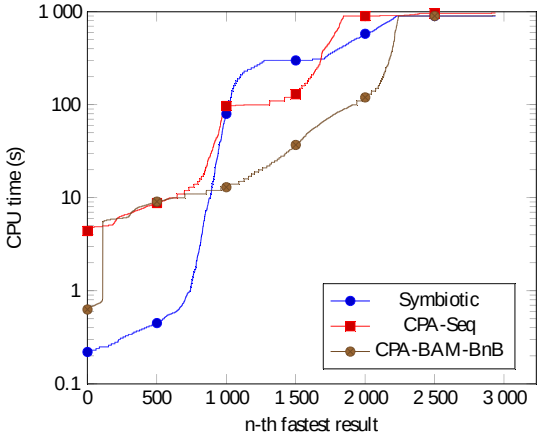


Fig. 4. Statistics for winners of SV-COMP'20 in the «Software Systems» category

Verification tools often implement algorithms sequentially because there is a considerable overhead to share complicated internal data structures. A few tools can use multi-core CPUs or distributed computing and there are several tools that employ GPUs. To substantially speed up solution of many independent verification tasks, verification tools are executed in parallel at IaaS or PaaS clouds and clusters. You can find more information about this in the appropriate survey [9].

6. Environment Modeling and Checking Program Fragments

Libraries, user inputs, other programs, etc. constitute an environment that can influence a program execution. To verify the program, it is necessary to provide a model which represents certain assumptions about the environment:

[3] Indeed, programs should carefully inspect return values of called functions if so since often they can represent error codes.

- The environment model should invoke a program API in a way the environment can do.
- It should contain models of undefined functions which the program calls during execution and which can influence verification results for checked requirements.

Bug finding is possible even without very accurate environment models. Still, more accurate environment models help to improve code coverage and reduce a false alarm rate. To achieve high-quality verification results it is crucial to provide the precise environment model taking into account specifics of checked requirements and programs under verification.

At environment modeling it is important to distinguish parallel and sequential cases. It is pretty natural to have a parallel environment model that can accurately reflect all possible interactions with the real environment. Unfortunately, as it was already mentioned, it may be too hard for verification. Thus, one has to provide sequential environment models and verify target programs with them. For instance, for libraries defining a set of functions it is possible to invoke them one by one[4]. For event-driven programs one can invoke callbacks just after their registration is completed or upon appropriate events are triggered by target programs [10].

To drastically reduce consumption of computational resources and increase chances to obtain verification results in a reasonable time, one can verify program fragments of a moderate size separately. A program fragment can contain several source files of the program and libraries, or just particular functions from them.

It becomes even more important to provide the appropriate environment model to avoid missing faults and false alarms at verification of such program fragments. Decomposing a program into individual C source files or even particular functions, which is an obvious way to simplify verification tasks, can require enormous efforts for modeling the environment. From common sense and from practical experience one needs to treat logically interconnected program components as program fragments like, say, loadable kernel modules or plugins (Table 2). Often this is a «golden mean» that enables obtaining useful verification results with moderate efforts for modeling the environment.

Table 2. Approximate number of components in open source projects

Project	Number of components
Linux kernel	5000
BusyBox	300
GTK library	200
Apache HTTP Server	150
VLC media player	80

The SV-COMP community does not propose any commonly accepted means for decomposing large programs into fragments and for specifying the environment model.

7. Formal Confirmation and Manual Analysis of Verification Results

After each successful run, verification tools provide proofs (*correctness witnesses*) and counterexamples (*violation witnesses*) in a machine-readable format [11]. The proposed witness validation technique establishes confirmation of such witnesses detecting spurious ones [12, 13]. The technique is widely used in SV-COMP, so today all verification tools participating in the competition can provide witnesses. This paper considers just violation witnesses since correctness witnesses serve primarily for validation and cannot help expert to comprehend proofs.

A violation witness describes a subset of paths from an entry point to a found error. By design, it can miss some details and even some parts of these paths. A witness validation tool considers the

[4] The same functions can be invoked multiple times, but still sequentially.

violation witness in combination with a control-flow automaton extracted from the program representation to recheck the solution of the verification task.

Although violation witnesses can be automatically validated, users still need to investigate them manually to understand reasons of faults and false alarms (false alarms can be caused by either imprecision of verification tools or environment models). There are some tools for visualization of witnesses in a more user-friendly way, but they do not help much for large programs because visualizations can contain too many details irrelevant for checking particular requirements. Moreover, experts need means for assessing verification results obtained for different versions and configurations of target software.

Some verification tools, e.g. CPAchecker [14], can provide code coverage reports in addition to witnesses [15]. These reports are in the GCC test coverage format (GCOV). For its visualization one can use standard tools like LCOV [16].

Code coverage reports are an important artifact to establish verification in practice. They reflect parts of the program such as lines of code, branches and functions that are actually verified. This information is essential for estimating an environment model quality since neither violation nor correctness witnesses do not provide data on actually considered program paths. Code coverage reports help to understand which program entry points should be invoked additionally by environment models. SV-COMP does not focus on this useful artifact.

8. Conclusion

Verification tools participating in SV-COMP demonstrate excellent results for verification tasks included into the benchmark suite. Some verification tools miss a few faults and report not so many false alarms. This works for moderate-sized programs that are prepared in advance and checked against the predefined list of properties.

Industrial C programs can contain much more source code than typical verification tasks. Also, during work they can extensively interact with their environments. This hinders or even makes impossible application of verification tools for them. Moreover, users can need to check specific requirements in addition to supported properties. At last, existing tools do not provide users with a comprehensive enough suite of means for analysis of verification results.

To reduce efforts that are necessary for application of verification tools for large industrial C programs one develops tools and infrastructures around them, e.g. SDV [17], Klever [18, 19] and VerifierCloud [20]. They provide very different capabilities and look very diversely. Often they are intended just for a particular type of programs and bound with the only verification tool. Thus, large-scale application of software model checkers still requires more research and development to cope with all their restrictions and get all expected benefits.

References

[1]. P.E. Black, L. Badger et al. Dramatically reducing software vulnerabilities: report to the White House Office of Science and Technology Policy. Technical Report NISTIR 8151, National Institute of Standards and Technology, Gaithersburg, MD, 2016.

[2]. G. Klein, J. Andronick et al. Comprehensive formal verification of an OS microkernel. ACM Transactions on Computer Systems, volume 32, issue 1, 2014, pp. 1-70.

[3]. D. Efremov, M. Mandrykin, A. Khoroshilov. Deductive verification of unmodified Linux kernel library functions. Lecture Notes in Computer Science, vol. 11245, 2018, pp. 216-234.

[4]. D. Beyer. Competition on software verification. Lecture Notes in Computer Science, vol. 7214, 2012, pp. 504-524.

[5]. D. Beyer. Software verification: 10th comparative evaluation (SV-COMP 2021). Lecture Notes in Computer Science, vol. 12652, 2021, pp. 401–422.

[6]. Definitions and Rules of 10th International Competition on Software Verification. URL: https://sv-comp.sosy-lab.org/2021/rules.php, accessed 12.12.2021.

[7]. V. Mutilin, E. Novikov, A. Khoroshilov. Analysis of typical faults in Linux operating system drivers. Trudy ISP RAN/Proc. ISP RAS, vol. 22, 2012, pp. 349-374 (in Russian). https://doi.org/10.15514/ISPRAS-2012-22-19.
[8]. P. Andrianov. Analysis of correct synchronization of operating system components. Programming and Computer Software, vol. 46, issue 8, 2020, pp. 712-730. https://doi.org/10.1134/S0361768820080022.
[9]. I. Zakharov. A survey of high-performance computing for software verification. Communications in Computer and Information Science, vol. 779, 2017, pp. 196-208.
[10]. I. Zakharov, E. Novikov. Compositional environment modelling for verification of GNU C programs. In: Proc. of the Ivannikov Ispras Open Conference, 2018, pp. 39–44.
[11]. Exchange Format for Violation Witnesses and Correctness Witnesses. URL: https://github.com/sosy-lab/sv-witnesses, accessed 12.12.2021.
[12]. D. Beyer, M. Dangl et al. Correctness witnesses: exchanging verification results between verifiers. In Proc. of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016, pp. 326-337.
[13]. D. Beyer, M. Dangl et al. Witness validation and stepwise testification across software verifiers. In Proc. of the 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 721-733.
[14]. D. Beyer, M.E. Keremoglu. CPAchecker: a tool for configurable software verification. Lecture Notes in Computer Science, vol. 6806, 2011, pp. 184–190.
[15]. R. Castaño, V. Braberman et al. Model checker execution reports. In Proc. of the 32nd IEEE/ACM International Conference on Automated Software Engineering, 2017, pp. 200-205.
[16]. LCOV – the LTP GCOV extension. URL: https://github.com/linux-test-project/lcov, accessed 12.12.2021.
[17]. T. Ball, V. Levin, S.K. Rajamani. A decade of software model checking with SLAM. Communications of the ACM, vol. 54, issue 7, 2011, pp. 68-76.
[18]. Klever: a software verification framework. URL: https://forge.ispras.ru/projects/klever, accessed 12.12.2021.
[19]. E. Novikov and I. Zakharov. Towards automated static verification of GNU C programs. Lecture Notes in Computer Science, vol. 10742, 2018, pp. 402-416.
[20]. VerifierCloud Web Interface. URL: https://vcloud.sosy-lab.org, accessed 12.12.2021.

Информация об авторах / Information about authors

Евгений Михайлович НОВИКОВ – кандидат физико-математических наук, ведущий научный сотрудник отдела Технологий программирования ИСП РАН. Его научные интересы включают верификацию моделей программ, спецификацию требований и операционные системы.

Evgeny Mikhailovich NOVIKOV – PhD, Senior Researcher at the Software Engineering Department of ISP RAS. His research interests include software model checking, requirements specification and operating systems.

DOI: 10.15514/ISPRAS-2021-33(6)-2

Мониторинг и тестирование модулей операционных систем на основе абстрактных моделей поведения системы

[1] *Д.В. Ефремов, ORCID: 0000-0002-9916-056X <efremov@ispras.ru>*
[1] *В.В. Копач, ORCID: 0000-0003-2461-9986 <vkopach@ispras.ru>*
[1,2] *Е.В. Корныхин, ORCID: 0000-0001-9303-3132 <kornevgen@ispras.ru>*
[1,2,3] *В.В. Кулямин, ORCID: 0000-0003-3439-9534 <kuliamin@ispras.ru>*
[1,2,3] *А.К. Петренко, ORCID: 0000-0001-7411-3831 <petrenko@ispras.ru>*
[1,2,3,4] *А.В. Хорошилов, ORCID: 0000-0002-6512-4632 <khorochilov@ispras.ru>*
[1,3] *И.В. Щепетков, ORCID: 0000-0002-5794-004X <shchepetkov@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, г. Москва, Ленинские горы, д. 1*
[3] *НИУ Высшая школа экономики, 101978, Россия, г. Москва, ул. Мясницкая, д. 20*
[4] *Московский физико-технический институт, 114701, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*

Аннотация. В связи с высокой сложностью современных операционных систем (ОС) для спецификации даже отдельных аспектов их функциональности, как, например, функций безопасности, приходится использовать достаточно сложные модели на высокоуровневых языках. При этом задача верификации соответствия таким моделям серьезно усложняется из-за необходимости установления связей между реализацией операционной системы и моделью, представленными на сильно отличающихся языках. В данной работе мы представляем методику установления и поддержания таких связей, которую можно эффективно использовать при тестировании и мониторинге операционных систем. Описанная методика была успешно применена при тестировании и мониторинге компонентов ядра операционной системы Linux с использованием моделей на Event-B.

Ключевые слова: верификация во время выполнения; ядро Linux; LSM; IMA/EVM; Event-B; ProB

Для цитирования: Ефремов Д.В., Копач В.В., Корныхин Е.В., Кулямин В.В., Петренко А.К., Хорошилов А.В., Щепетков И.В. Мониторинг и тестирование модулей операционных систем на основе абстрактных моделей поведения системы. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 15-26. DOI: 10.15514/ISPRAS–2021–33(6)–2

Благодарности: Данная работа выполнена при поддержке гранта РФФИ 20-01-00568.

Runtime Verification of Operating Systems Based on Abstract Models

[1] *D.V. Efremov, ORCID: 0000-0002-9916-056X <efremov@ispras.ru>*
[1] *V.V. Kopach, ORCID: 0000-0003-2461-9986 <vkopach@ispras.ru>*
[1,2] *E.V. Kornykhin, ORCID: 0000-0001-9303-3132 <kornevgen@ispras.ru>*
[1,2,3] *V.V. Kuliamin, ORCID: 0000-0003-3439-9534 <kuliamin@ispras.ru>*
[1,2,3] *A.K. Petrenko, ORCID: 0000-0001-7411-3831 <petrenko@ispras.ru>*
[1,2,3,4] *A.V. Khoroshilov, ORCID: 0000-0002-6512-4632 <khorochilov@ispras.ru>*
[1,3] *I.V. Shchepetkov, ORCID: 0000-0002-5794-004X <shchepetkov@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
[2] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia.
[3] *National Research University, Higher School of Economics*
20, Myasnitskaya ulitsa, Moscow, 101978, Russia.
[4] *Moscow Institute of Physics and Technology (MIPT),*
9, Campus per., Dolgoprudny, Moscow region, 114701, Russia.

Abstract. High complexity of a modern operating system (OS) requires to use complex models and high-level specification languages to describe even separated aspects of OS functionality, e.g., security functions. Use of such models in conformance verification of modeled OS needs to establish rather complex relation between elements of OS implementation and elements of the model. In this paper we present a method to establish and support such a relation, which can be effectively used in testing and runtime verification/monitoring of OS. The method described was used successfully in testing and monitoring of Linux OS core components on conformance to Event-B models.

Keywords: runtime verification; linux kernel; LSM; IMA/EVM; Event-B; ProB

For citation: Efremov D.D., Kopach V.V., Kornykhin E.V., Kuliamin V.V., Petrenko A.K., Khoroshilov A.V., Shchepetkov I.V. Runtime Verification of Operating Systems Based on Abstract Models. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 15-26 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–2

Acknowledgements. This work was supported by the RFBR grant No. 20-01-00568.

1. Введение

Сложность современных операционных систем (ОС) требует соответствующей поддержки со стороны методов верификации и валидации, использующихся при их разработке. Одним из многообещающих методов, способных справиться с возрастающей сложностью, является использование моделей, которые абстрагируются от многих низкоуровневых и специфичных деталей реализации ОС. Также помогает разделение отдельных аспектов функциональности, безопасности, производительности и др. по разным уровням или модулям моделей. Объединенные вместе, данные методы значительно упрощают процессы задания и проверки требований, а также использование различных инструментов анализа и доказательства.

Практика показывает, что для точного и полного описания таких аспектов современных индустриальных ОС, как, например, контроль доступа, нужны сложные модели, разработанные с использованием достаточно выразительных языков. Мы успешно использовали Event-B [1] в нескольких проектах для описания и формальной верификации моделей контроля доступа в ОС, основанных на Linux, включая реализации мультиуровневых защитных механизмов [2, 3]. Следующим шагом является использование полученных моделей для проверки их соответствия реальному поведению ОС с целью выявления возможных расхождений и противоречий.

Учитывая многочисленные сложности, связанные с эффективным применением подходов дедуктивной верификации при работе с кодом ядра ОС, подходящими для выполнения данной задачи кажутся методы тестирования и динамической верификации [4]. Для этого требуется разработать метод тестирования и мониторинга с использованием сравнительно большой Event-B модели в качестве оракула. Статья рассматривает применение формальных моделей в мониторинге и динамической верификации поведения ОС и содержит описание ключевых аспектов предлагаемого метода.

2. Event-B модели

Event-B [1] – это формальный метод, основанный на теории множеств и логике предикатов.

Он имеет простую нотацию и поддерживается платформой для разработки и верификации моделей Rodin [5]. Модель на языке Event-B – это *дискретная система переходов*, в которой компоненты моделируемой системы описываются с помощью *состояний*, переходов между состояниями, которые называются *событиями*, и свойствами, которые должны соблюдаться в каждом состоянии (*инварианты*). Модели можно декомпозировать на отдельные модули, называемые уровнями, с помощью техники *пошагового уточнения* (stepwise refinement) [6].

Rodin способен автоматически генерировать *условия верификации*: утверждения, которые должны быть доказаны, чтобы демонстрировать корректность модели. Rodin также предоставляет средства для их автоматического и интерактивного доказательства. В качестве расширения для Rodin, а также в виде независимого инструмента, доступен инструмент анимации и проверки моделей ProB [7].

Мы используем Event-B для создания моделей контроля доступа ядер ОС, основанных на Linux. Эти модели могут быть довольно абстрактны: в таком случае они называются моделями *политик безопасности*; или же они могут быть более полными и ближе к реализации ОС: такие модели называются *функциональными спецификациями*. Обычно мы создаем модели обоих типов для каждой ОС, с которой приходится работать. Это позволяет облегчить анализ свойств безопасности и доказательство их непротиворечивости, в то же время сохраняет возможность демонстрации соответствия моделей с кодом (реализацией). Мы описали процесс работы с несколькими моделями разного уровня абстракции и доказательством их соответствия в работе [8].

Функциональные спецификации достаточно являются низкоуровневыми и описывают *системные вызовы* ядра Linux. Мы предлагаем использовать формальные спецификации как оракулы в тестировании для проверки того, что результаты выполнения системных вызовов не ведут к нарушению требований политик безопасности.

3. Контроль доступа в Linux

Ядро Linux, начиная с версии 2.5 (2002 год) [9], включает в свой состав подсистему Linux Security Modules (LSM). Данная подсистема представляет собой набор модулей, усиливающих безопасность ядра и реализующих управление контролем доступа. Взаимодействие между модулями и ядром организовано с помощью API интерфейса LSM.

Этот API позволяет реализовывать дополнительные проверки доступа, усиливающие существующие. Он не предназначен для изменения состояния ядра, за пределами специально отведенных для этого интерфейса полей в структурах данных ядра.

Таким образом, побочный эффект модулей безопасности, корректно использующих этот интерфейс с архитектурной точки зрения, сводится только к досрочному завершению обработки какого-либо события в ядре. Ровно таким же образом работает механизм уточнений в Event-B.

В настоящий момент (ядро Linux версии 5.15) интерфейс представляет собой список из 238 (см. LSM_HOOK в include/linux/lsm_hook_defs.h) возможных типов обработчиков событий в ядре. Модуль безопасности при инициализации явным образом регистрирует собственные обработчики, которые необходимы ему для реализации заложенных в нём политик безопасности. Ядро Linux в процессе своей работы и при наступлении определенных условий вызывает зарегистрированные обработчики либо для оповещения о каком-то из событий, например, о выделении нового индексного дескриптора в файловой подсистеме, либо для контроля операции со стороны модулей безопасности. В последнем случае в зависимости от результата, который возвращает модуль безопасности, обработка операции может быть либо продолжена ядром, либо завершена с кодом ошибки запрета доступа.

LSM интерфейс постоянно меняется в процессе разработки ядра, туда добавляются как новые контролирующие и оповещающие функции, так и убираются неиспользуемые и избыточные операции (см. рис. 1).

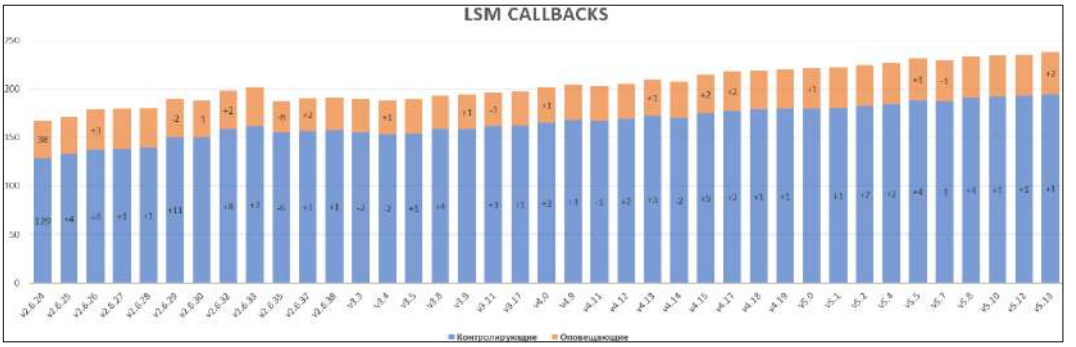


Рис. 1. Изменения LSM интерфейса в ядре Linux
Fig. 1. LSM interface changes in the Linux kernel

Изначально несколько LSM модулей не могли работать совместно. Позднее, разработчики ядра доработали интерфейс [10], чтобы каждому LSM обработчику соответствовал список в ядре. В таком случае ядро вызывает все зарегистрированные обработчики LSM по списку, в соответствии с приоритетом регистрации их модулей безопасности в ядре. Это дает возможность включать один основной (major) модуль безопасности, который прикрепляет собственные данные и метки к структурам ядра, и несколько вспомогательных (minor), которые не требуют хранения дополнительных данных для реализации политик безопасности. Постепенно разработчики ядра дорабатывают интерфейс [11] и структуры ядра с той целью, чтобы хранимые модулями безопасности данные также отделялись друг от друга, что позволит включать несколько основных модулей одновременно. Приоритет вызова этих модулей по-прежнему останется.

Модуль IMA/EVM (Integrity Measurement Architecture and Extended Verification Module) контроля целостности Linux также представляет собой модуль безопасности LSM, но его проверки всегда вызываются последними. Так сложилось исторически из-за необходимости обеспечивать одновременную работу модуля контроля целостности с одним из модулей мандатного контроля (SELinux, Apparmor, Smack, Tomoyo) до того, как интерфейс LSM позволил включать несколько модулей одновременно. На текущий момент это также остается актуальным, так как проверки целостности (IMA) требуют сверки контрольной суммы, записанной в расширенных атрибутах файла, с контрольной суммой содержимого файла. Подсчет контрольных сумм является затратной операцией и её рационально откладывать до того момента, когда все остальные проверки уже выполнены. Модуль также позволяет контролировать целостность не только содержимого файла (IMA), но и его метаданных (EVM). Под метаданными в данном случае подразумеваются атрибуты файла,

расширенные атрибуты, права, владелец, группы. Модуль также поддерживает проверку криптографической подписи контрольных сумм.

IMA/EVM имеет несколько режимов работы. Для понимания принципов работы модуля важны только два из них. В первом для файлов высчитываются контрольные суммы, которые затем записываются в расширенные атрибуты. Проверок доступа при этом не выполняется. Во втором режиме выполняется сверка контрольных сумм при доступах к файлу на открытие, запись, выполнение. В случае их несовпадения доступ не предоставляется. Эти режимы не переключаются динамически: ядро ОС должно загрузиться в одном из них.

Важной особенностью LSM интерфейса является то, что ядро вызывает зарегистрированные обработчики только после базовых проверок параметров и после дискреционных проверок доступа. Политику дискреционного доступа отменить нельзя, и реализуется она не в отдельном модуле безопасности, а самим ядром.

Таким образом, на текущий момент контроль доступа в ядре Linux при обработке системного вызова выглядит следующим образом. Сначала выполняется базовая проверка аргументов системного вызова на корректность, далее самим ядром осуществляется проверка привилегий (capabilities), если это подразумевается системным вызовом и его аргументами (например, работа с сырыми сокетами требует привилегии CAP_NET_RAW у процесса), затем также самим ядром выполняется дискреционная проверка прав, после этого ядром вызываются модули контроля доступа LSM в порядке приоритета их регистрации. После того как все LSM модули разрешили доступ, ядро обрабатывает системный вызов и возвращает результат. Если на каком-то из этапов доступ запрещается, то ядро досрочно завершает обработку системного вызова и возвращает процессу, который его инициировал, код ошибки (например, -EPERM или -EACCES). Как можно видеть, реализация контроля доступа в ядре не монолитна, а состоит из базовой неотключаемой части и из дополнительных модулей, которые только могут вводить лишь новые проверки, ограничивающие доступ, никак не влияя на результаты прошлых проверок. Такое разделение хорошо ложится на Event-B модели и механизм уточнений.

Модули безопасности явно полагаются на то, что ядро корректным образом вызывает функции LSM интерфейса во время обработки событий. Политики безопасности, которые реализуются этими модулями, подразумевают, что LSM интерфейс достаточен (содержит необходимые контролирующие обработчики) для их реализации в виде модулей безопасности. При динамической верификации мы проверяем не только то, что модуль безопасности реализует политику безопасности корректным образом, но также то, что ядро вызывает модуль безопасности во время обрабатываемых событий и достаточность самого LSM интерфейса для реализации требуемой политики безопасности.

С точки зрения выявления дефектов реализации политик безопасности наиболее приоритетны расхождения вида "модель запрещает"<==>"ядро ОС разрешает". На практике расхождения, связанные с нехваткой физических ресурсов, крайне редки, а большинство выявляемых расхождений связано с тем, что модель недостаточно полно отражает поведение реальной системы. Многие пограничные случаи не описаны в официальной документации системных вызовов и выявляются на практике в виде расхождений.

4. Динамическая верификация

Далее, под моделью понимается функциональная спецификация поведения системных вызовов ядра ОС на языке Event-B.

В общем виде, схема динамической верификации выглядит следующим образом.

1) Подготавливается начальное окружение для теста. Сюда входят файлы, директории, ссылки, владельцы и группы, их права доступа, расширенные атрибуты модуля мандатного контроля доступа и модуля контроля целостности и другие данные,

необходимые для запуска теста.

2) Снимаются данные начального окружения, подготовленные на прошлом шаге, как и данные самого файла теста. Они записываются в базу данных.
3) Осуществляется контролируемое тестовое воздействие на ядро ОС в изолированной среде.
4) Параллельно с прошлым шагом в базу данных записываются результаты обработки системных вызовов ядром ОС с включенными модулями безопасности.
5) Собранные данные о начальном окружении теста и результатах обработки системных вызовов ядром ОС преобразовываются в вид (трассу), пригодный для воспроизведения на модели.
6) Запускается процесс анимации модели с использованием инструмента ProB.
7) Состояние модели наполняется данными, снятыми с начального окружения теста на шаге 2 и преобразованными на шаге 5.
8) Системные вызовы с их аргументами и результатами, снятые на шаге 4 и преобразованные на шаге 5, анимируются на модели. В случае расхождения результатов реальных операций и модельных операций воспроизведение трассы прекращается, результаты о причинах расхождения записываются в журнал тестирования. В некоторых случаях расхождение не критично. Тогда анимация откатывается до состояния прошлого системного вызова и продолжается с пропуском системного вызова, на котором возникло расхождение. Процесс анимации завершается успехом, когда все системные вызовы были воспроизведены на модели и результаты их обработки ядром ОС соответствуют модельным.
9) Процесс повторяется для каждого теста.

Шаги 1-4 и шаги 5-8 могут быть разнесены по времени и выполняться отдельно друг от друга на разных машинах.

На шаге 8 возможны ситуации, когда результаты обработки системных вызовов совпадают и когда они не совпадают. В первом случае процесс анимации продолжается.

Лишь в случае, когда модель запрещает доступ, а ядро операционной системы запрещает доступ с кодом ошибки, не связанным с его контролем (например, код нехватки памяти - ENOMEM), в журнал теста заносится предупреждение. В ситуации, когда модель запрещает доступ, а ядро ОС его разрешило, анимация прекращается, в журнал теста заносятся ошибка, состояние модели, нарушенное в модели условие.

Обратная ситуация (модель разрешает, ядро ОС запрещает) также может быть связана с ошибкой реализации политики безопасности. Но она менее критична с точки зрения безопасности, поэтому в таком случае процесс анимирования продолжается с пропуском системного вызова, на котором произошло расхождение. Если в реальной системе доступ был запрещен по причине нехватки ресурсов, например, код ошибки -ENOMEM, то событие просто пропускается при воспроизведении. Если же код ошибки связан с контролем доступа (например, -EPERM или -EINVAL), то это может свидетельствовать о некритичной ошибке в реализации, приводящий к запрету доступа там, где это не требуется с точки зрения политики безопасности. Информация о таком расхождении заносится в журнал теста, воспроизведение продолжается.

4.1 Реализация

Для реализации динамической верификации были разработаны компоненты мониторинга системных вызовов в ядре ОС, сбора информации о начальном окружении теста, трансляции трасс, а также компонент воспроизведения трасс на модели.

Компонент сбора информации о начальном окружении для теста собирает данные о пользователях в системе, их группах, информацию о файлах теста, передаваемых ему в качестве аргументов. С файлов и самого теста снимается информация о дискретных правах, владельце, группе, индексном дескрипторе, томе монтирования, полная информация о файлах и директориях до корневого раздела.

Монитор системных вызовов реализован в виде модуля ядра Linux и программы пользовательского пространства. Модуль мониторинга использует механизмы ядра Linux kprobes, kretprobes для перехвата аргументов системных вызовов и результатов их обработки, а также сокеты netlink для отправки сообщений с событиями системных вызовов программе пользовательского пространства. Последняя управляет процессом мониторинга и записывает его результаты в базу данных.

Компонент трансляции трасс осуществляет преобразование трасс системных вызовов с реальной системы в трассу для воспроизведения на модели. В качестве аргументов использует данные о начальном окружении теста, а также трассу, которая ранее была собрана при мониторинге. При трансляции трассы все реальные сущности транслируются в номерные элементы состояния модели. В модельную трассу также добавляются дополнительные события, которые соответствуют шагам подготовки начального окружения из тестов. Например, добавляется событие init, позволяющее за один раз обновить состояние модели до соответствия начальному окружению теста. Если бы его не было, требовалось бы реконструировать последовательность системных вызовов, необходимую для достижения этого состояния модели. Также, например, в спецификацию Event-B добавляется дополнительное псевдособытие login, которое позволяет в трассе отделить подготовительные действия с моделью от начала воспроизведения самого теста.

Компонент анимации осуществляет воспроизведение трасс на модели и сверку результатов доступов, полученных на реальной системе, на соответствие модели. В конце работы выводит статистику покрытия элементов модели и результат тестирования, нарушенные в модели условия, в случае наличия таковых. Компонент реализован на основе библиотеки ProB de.prob2.kernel.

4.2 Тесты

Далее мы опишем тесты, используемые в рамках динамической верификации. Тест – это программа, осуществляющая тестовое воздействие на тестируемую ОС. Тестовым воздействием является один из системных вызовов ядра Linux, включенных в функциональную спецификацию. С тестом связано описание состояния ОС перед тестовым воздействием (какие должны существовать файлы, директории, пользователи и группы, метки и т.п.). Это описание вместе с целевым системным вызовом хранится в базе данных и используется для подготовки начального окружения.

Основой для построения тестового набора были функциональные требования системных вызовов, представленные в модели. Альтернативой было бы использование самой модели, ее структуры, для построения тестового набора. Использование требований позволяет при помощи тестового набора проверить не только реализацию ОС, но и модель.

Апробация подхода проводилась на модели системных вызовов, взаимодействующих с файловой системой. Список этих системных вызовов такой: open, creat, mkdir, getdents, getxattr, setxattr, link, symlink, chmod, chown, execve, unlink.

Функциональные требования были поделены на 3 категории: свойства дискретного контроля доступа (DAC), свойства мандатного контроля доступа (MAC) и свойства целостности (IMA/EVM) [12]. Каждая категория имеет свои подкатегории, например, доступа к объекту, обхода пути и т.д. Тестами покрывается доступ для каждой из категорий для каждого системного вызова для каждого типа объекта (файл, директория, ссылка) для каждого типа

пользователя (привилегированный, непривилегированный, со специальными возможностями) для каждого режима системного вызова (если применимо).

Тесты минималистичны, независимы и воспроизводимы: они выполняют только один системный вызов под мониторингом. Данный подход позволяет упростить отладку ядра ОС вместе с системой контроля доступа и выявлять неточности/неполноту в функциональной спецификации. Также подход упрощает трансляцию тестовой трассы в модельную и упрощает процесс воспроизведения трассы на модели.

Табл. 1. Количество тестов
Table 1. Number of tests

syscall	**DAC**	**MAC**	**IMA/EVM**	**all**
open	141	14084	45	14270
Creat	57	9268	15	9340
mkdir	57	9268	15	9340
getdents	0	26	0	26
getxattr	110	9372	30	9512
setxattr	110	9372	30	9512
link	57	9268	15	9340
symlink	55	9268	15	9338
chmod	110	9372	30	9512
chown	110	9372	30	9512
execve	55	4712	15	4782
unlink	3015	27804	15	30834
all	3877	121186	255	125318

Используется комбинаторный подход для генерации тестов. Тесты DAC проверяют доступ к объекту (файлу, директории), обход пути до объекта, директории с установленными битами SetGID и Sticky. В тестах перебираются тип объекта (файл/директория), системный вызов, владельцы объекта, пользователь и группа субъекта доступа (процесса), списки ACL, дополнительные группы субъекта, длина пути до объекта и права доступа директорий в пути.

Тесты MAC проверяют мандатные правила доступа к объекту, к его родительской директории, доступа к объекту при наличии специальных атрибутов. В этих тестах перебираются тип объекта (файл/директория), мандатная метка объекта и субъекта (процесса), системный вызов, владелец объекта, владелец субъекта, длина пути к объекту, мандатные метки директорий в пути к объекту, специальные атрибуты.

В тестах подсистемы целостности IMA/EVM варьируются следующие параметры тестов:

- подпадает ли объект доступа под контроль целостности;
- атрибут неизменности (immutable bit) объекта доступа;
- контрольная сумма файла, хранящаяся в расширенных атрибутах файла.

Для минимизации тестового набора не генерируются все пары мандатных меток для субъектов и объектов. Вместо этого используются только наборы пар из меток, находящихся в различных важных соотношениях – метки эквивалентны, первая доминирует, вторая доминирует, метки несравнимы.

По итогу воспроизведения трассы на модели печатается статистика покрытия элементов модели. Планируется использовать эту информацию для дополнения тестового набора для более полного покрытия элементов модели и частей условий в модели.

5. Результаты

В рамках данной работы был реализован прототип системы динамической верификации, сверяющий поведение ядра Linux вместе с закрытым модулем мандатного контроля доступа

и модулем контроля целостности IMA/EVM с абстрактными моделями поведения системы на языке Event-B, формализующих политики безопасности и требования контроля доступа к операционной системе в целом. Прототип был апробирован на наборе системных вызовов Linux, связанных с доступом к файловой системе.

Это позволило выявить ошибки в модели системного вызова setxattr, когда разрешались только флаги XATTR_CREATE и XATTR_REPLACE и не учитывалось, что переменная флага может не иметь специального значения и быть равной нулю. Также модель избыточно требовала наличия прав доступа на чтение и на выполнение к исполняемому файлу при системном вызове execve. Ядро операционной системы наличие права на чтение не проверяет. В модели некорректно проверялись права дискреционного доступа на запись в каталоге, в котором пользователь создает ссылку на файл, что приводило к ситуации, когда доступ запрещался в реальной системе, но разрешался моделью.

В поставляемой вместе с модулем мандатного контроля доступа утилите редактирования меток мандатного контроля доступа была найдена ошибка, проявлявшаяся в некорректном назначении прав доступа на файл при использовании флага игнорирования части проверок.

В реализации модуля мандатного контроля доступа была выявлена логическая ошибка, связанная с тем, что при создании ссылок не проверялись права доступа к директории файла, на который создается ссылка. Это приводило к ситуации, когда ссылка успешно создавалась, чего не могло происходить с точки зрения политики безопасности. Также в реализации модуля контроля целостности метаданных EVM была выявлена ошибка, связанная с тем, что метки безопасности мандатного контроля доступа не измерялись (не вычислялась контрольная сумма). Изменение мандатной метки должно было привести к выявлению нарушения целостности метаданных файла, чего в реализации не происходило.

6. Аналогичные работы

В работе [13] рассматривается вопрос корректности вызовов функций LSM интерфейса ядром, нет ли путей в коде, в которых пропущены эти вызовы. Авторы используют тот факт, что в большинстве случаев вызовы LSM функций в ядре расставлены корректно. Они собирают трассы системных вызовов ядра, их аргументов, результаты LSM функций и функций доступа к основным структурам данных ядра. На основе собранной информации выявляются аномалии, когда в большинстве случаев при системном вызове LSM интерфейс контролирует доступ к данным ядра, но есть пути в коде ядре, где требуемые проверки отсутствуют.

В продолжении [14] своей прошлой работы авторы проверяют не только необходимость LSM вызовов на определенных путях исполнения в ядре Linux, но и их достаточность. В прошлой работе авторы делали предположение о том, что в большей части ядра необходимые LSM вызовы уже расставлены корректным образом. В данной работе авторы моделируют необходимые LSM вызовы при определенных типах доступа к структурам данных ядра. Выявляются дополнительные ошибки, когда LSM вызов есть, но часть данных им не контролируется и не проверяется, что может приводить к изменению критических структур данных без соответствующих проверок и к полному игнорированию их в будущем за счет неавторизованного повышения привилегий. Авторами используется тот же подход с динамическим сбором трасс выполнения с ядра Linux и их последующим статическим анализом.

В работе [15] используется динамическая верификация для анализа модулей ядра Linux, обсуждаются подходы к инструментации загруженных модулей ядра Linux и представляется инструмент динамической верификации KEDR. Данный инструмент позволяет перехватывать выполнение функций, отслеживать утечки памяти, имитировать (fault injection) коды ошибок некоторых функций ядра для проверок путей их обработки в модулях.

Работа [16] использует автоматы для анализа состояний потоков ядра в рамках ядра реального времени PREEMPT_RT Linux. Важным отличием от остальных работ является то, что весь анализ производится в самом ядре, так как частота перехватываемых событий настолько велика, что невозможно производить запись о них в журнал событий, записывать его из ядра в файл и анализировать после. Весь анализ производится непосредственно в момент наступления событий в ядре. Авторы описывают несколько небольших автоматных моделей состояний ядра, транслируют их в код на языке Си, собирают в модули ядра и прикрепляют события переходов из одного состояния автомата в другое к вызовам функций ядра с помощью механизмов трассировки Linux. Самого факта вызова определенных функции без анализа его аргументов и контекста его вызова достаточно в рамках рассматриваемых автоматных моделей.

7. Заключение

В данной работе мы представляем метод проведения тестирования и мониторинга компонентов ядра ОС на основе спецификаций части функциональности, зафиксированной в формальной модели на языке Event-B. Описанный метод позволяет эффективно верифицировать реализацию ОС на соответствие достаточно сложным формальным моделям, представленным на языке, отличном от языка реализации, при помощи установления соответствия между событиями модели и системными вызовами, а также переменными модели и специфическими данными, получаемыми из трассировки параметров системных вызовов. Особенностью метода является построение достаточно сложной исходной конфигурации ОС, в рамках которой становится возможно реализовать необходимое разнообразие тестовых ситуаций, покрывающих разнообразные условия, присутствующие в спецификациях событий в формальной модели.

Метод был успешно применен для тестирования нескольких разных реализаций модуля ядра ОС Linux LSM, отвечающего за выполнение дополнительных к традиционным для Linux процедур контроля доступа. В тестируемых компонентах LSM были реализованы правила многоуровнего или мандатного управления доступом.

Список литературы / References

[1]. J.-R. Abrial. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010, 612 p.

[2]. P.N. Devyanin, A.V. Khoroshilov et al. Using Refinement in Formal Development of OS Security Model. Lecture Notes in Computer Science, vol. 9609, 2016, pp. 107-115.

[3]. V. Kuliamin, A. Khoroshilov и D. Medveded. Formal Modeling of Multi-Level Security and Integrity Control Implemented with SELinux. In Proc. of the International Conference on Actual Problems of Systems and Software Engineering (APSSE), 2019, pp. 131-136.

[4]. А.К. Петренко, Д.В. Ефремов и др. Мониторинг и тестирование на основе многоуровневых спецификаций программ. Труды ИСП РАН, том 32, вып. 6, 2020 г., стр. 7-18 / A.K. Petrenko, D.V. Efremov et al. Monitoring and testing based on multi-level program specifications. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 6, 2020, pp. 7-18 (in Russian). DOI: 10.15514/ISPRAS–2020–32(6)–1.

[5]. J.-R. Abrial, M. Butler et al. Rodin: an open toolset for modelling and reasoning in Event-B. International Journal on Software Tools for Technology Transfer, vol. 12, issue 6, 2010, pp. 447-466.

[6]. N. Wirth. Program Development by Stepwise Refinement. Communications of the ACM, vol. 14, issue 4, 1971, pp. 221-227.

[7]. M. Leuschel, M. Butler. ProB: A Model Checker for B. Lecture Notes in Computer Science, vol. 2805, 2003, pp. 855-874.

[8]. A. Khoroshilov, V. Kuliamin et al. A State-based Refinement Technique for Event-B. In Proc. of the Ivannikov Memorial Workshop (IVMEM), 2020, pp. 49-54.

[9]. G. Kroah-Hartman. LSM: Add all of the new security/* files for basic task control. URL: https://git.kernel.org/pub/scm/linux/kernel/git/tglx/history.git/commit?id=2b15fe6334aebd7d3340f8b826acb79b138afa74, accessed 15.01.2022.

[10]. J. Edge. Progress in security module stacking. URL: https://lwn.net/Articles/635771/, accessed 15.01.2022.
[11]. J. Edge. LSM stacking and the future. URL: https://lwn.net/Articles/804906/, accessed 15.01.2022.
[12]. Linux Integrity Subsystem. URL: http://linux-ima.sourceforge.net/, accessed 15.01.2022.
[13]. A. Edwards, T. Jaeger, X. Zhang. Runtime verification of authorization hook placement for the Linux security modules framework. In Proc. of the 9th ACM Conference on Computer and Communications Security, 2002, pp. 225-234.
[14]. T. Jaeger, A. Edwards, X. Zhang. Consistency analysis of authorization hook placement in the Linux security modules framework. ACM Transactions on Information and System Security (TISSEC), vol. 7, issue 2, 2004, pp. 175-205.
[15]. V.V. Rubanov, E. A. Shatokhin. Runtime verification of linux kernel modules based on call interception. In Proc. of the Fourth IEEE International Conference on Software Testing, Verification and Validation, 2011, pp. 180-189.
[16]. D.B. de Oliveira, T. Cucinotta, R.S. de Oliveira. Efficient formal verification for the Linux kernel. Lecture Notes in Computer Science, vol. 11724, 2019, pp. 315-332.

Информация об авторах / Information about authors

Денис Валентинович ЕФРЕМОВ – научный сотрудник. Сфера научных интересов: формальная верификация, статический и динамический анализ.

Denis Valentinovich EFREMOV – researcher. Research interests: formal verification, static and dynamic analysis.

Виктория Владимировна КОПАЧ – научный сотрудник. Сфера научных интересов: методы верификации и валидации, анализ полноты требований.

Viktoria Vladimirovna KOPACH – researcher. Research interests: verification and validation methods, requirements completeness analysis.

Евгений Валерьевич КОРНЫХИН – кандидат физико-математических наук, доцент кафедры системного программирования МГУ, старший научный сотрудник ИСП РАН. Область интересов: формальная дедуктивная верификация моделей, тестирование на основе моделей.

Eugeny Valerievich KORNYKHIN – Ph.D. in Physics and Mathematics, Associate Professor of system programming departments at MSU and Senior Researcher at ISP RAS. Fields of Interest: formal deductive verification, model-based testing.

Виктор Вячеславович КУЛЯМИН – кандидат физико-математических наук, ведущий научный сотрудник ИСП РАН, доцент кафедр системного программирования МГУ и ВШЭ. Область интересов: формальная дедуктивная верификация моделей, тестирование на основе моделей.

Viktor Vyacheslavovich KULIAMIN – Ph.D. in Physics and Mathematics, leading researcher at ISP RAS, associate professor of system programming departments at MSU and the HSE. Fields of Interest: formal deductive model verification, model-based testing.

Александр Константинович ПЕТРЕНКО – доктор физико-математических наук, профессор, заведующий отдела ИСП РАН, профессор МГУ и ВШЭ. Область интересов: формальные методы программной инженерии, тестирование программного и аппаратного обеспечения, формальная спецификация требований.

Alexander Konstantinovich PETRENKO – Doctor of Physical and Mathematical Sciences, Professor, Head of the Department of ISP RAS, Professor of MSU and HSE. Areas of interest: formal methods of software engineering, testing of software and hardware, formal specification of requirements.

Алексей Владимирович ХОРОШИЛОВ – кандидат физико-математических наук, ведущий научный сотрудник, директор Центра верификации ОС Linux в ИСП РАН, доцент кафедр системного программирования МГУ, ВШЭ и МФТИ. Основные научные интересы: методы

проектирования и разработки ответственных систем, формальные методы программной инженерии, методы верификации и валидации, тестирование на основе моделей, методы анализа требований, операционная система Linux.

Alexey Vladimirovich KHOROSHILOV – Ph.D. in Physics and Mathematics, Leading Researcher, Director of the Linux OS Verification Center at ISP RAS, Associate Professor of System Programming Departments at MSU, HSE, and MIPT. Main research interests: design and development methods for critical systems, formal methods of software engineering, verification and validation methods, model-based testing, requirements analysis methods, Linux operating system.

Илья Викторович ЩЕПЕТКОВ – младший научный сотрудник. Область интересов: разработка и верификация формальных моделей компьютерных систем.

Ilya Viktorovich SHCHEPETKOV – Junior Researcher. Fields of Interest: development and verification of formal models of computer systems.

DOI: 10.15514/ISPRAS-2021-33(6)-3

Модель и декларативный язык спецификации бинарных форматов данных

[1] *А.А. Евгин, ORCID: 0000-0001-8826-6898 <evgin@ispras.ru>*
[1,2] *М.А. Соловьев, ORCID: 0000-0002-0530-6442 <icee@ispras.ru>*
[1,2] *В.А. Падарян, ORCID: 0000-0001-7962-9677 <vartan@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1

Аннотация. Ряд задач, связанных с бинарными форматами данных, включает в себя задачи разбора, генерации и совместного анализа кода и данных. Ключевым элементом для решения всех этих задач является универсальная модель формата. В данной работе предлагается подход к моделированию бинарных форматов. Описанная модель обладает достаточной выразительностью для спецификации большинства распространенных форматов. Отличительной особенностью модели является гибкость при задании положений полей, а также возможность описывать внешние поля, структура которых не определяется при разборе. В реализованной инфраструктуре имеется возможность создания и модификации представления с помощью программных интерфейсов. Предлагается алгоритм для разбора данных по модели, основаный на понятии вычислимости полей. В работе также представлен предметно-ориентированный язык спецификации форматов. Указываются описанные форматы и потенциальные практические применения модели для программного анализа форматированных данных.

Ключевые слова: бинарные форматы данных; декларативное описание; анализ бинарного кода; совместный анализ кода и данных

Для цитирования: Евгин А.А., Соловьев М.А., Падарян В.А. Модель и декларативный язык спецификации бинарных форматов данных. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 27-50. DOI: 10.15514/ISPRAS–2021–33(6)–3

Model and declarative specification language of binary data formats

[1] *A.A. Evgin, ORCID: 0000-0001-8826-6898 <evgin@ispras.ru>*
[1,2] *M.A. Solovev, ORCID: 0000-0002-0530-6442 <icee@ispras.ru>*
[1,2] *V.A. Padaryan, ORCID: 0000-0001-7962-9677 <vartan@ispras.ru>*
[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. A number of tasks related to binary data formats include the tasks of parsing, generating and conjoint code and data analysis. A key element for all of these tasks is a universal data format model. This paper proposes an approach to modeling binary data formats. The described model is expressive enough to specify the most common data formats. The distinctive feature of the model its flexibility in specifying field locations, as well as the ability to describe external fields, which do not resolve into detailed structure during parsing. Implemented infrastructure allows to create and modify a model using application programming interfaces. An

algorithm is proposed for parsing binary data by a model, based on the concept of computability of fields. The paper also presents a domain-specific language for data format specification. The specified formats and potential applications of the model for programmatic analysis of formatted data are indicated.

Keywords: binary data formats; declarative formats specification; binary code analysis; conjoint code and data analysis

For citation: Evgin A.A., Solovev M.A., Padaryan V.A. Model and declarative specification language of binary data formats. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 27-50 (in Russian). DOI: 10.15514/ISPRAS–2020–33(6)–3

1. Введение

Необходимость обработки сложных структур данных возникает во всех сферах системного и прикладного программирования. Форматы, используемые для хранения и передачи данных, обычно описываются человекочитаемыми спецификациями, которые могут быть как общедоступными, так и закрытыми. Говоря о форматах данных, мы обычно подразумеваем наличие некоторого объекта со своими характеристиками, который необходимо представить в адресном пространстве (например, в памяти или в сетевом потоке). Форматированными данными мы называем данные, снабженные разметкой, содержащей информацию о структуре данных и сопоставляющей фрагменты данных с характеристиками объекта (полями формата).

Ряд задач, связанных с форматами данных, достаточно разнообразен и возглавляется основной задачей разбора данных. Для обработки данных определенного формата программисту необходимо реализовать средство разбора, которое будет восстанавливать структуру данных и характеристики представленного объекта. Подходы к решению этой задачи очень разнообразны и варьируются от ручной реализации программы разбора конкретного формата до реализации системы автоматической генерации разборщиков по спецификации.

С учетом быстрого роста количества разнообразных форматов, хорошо зарекомендовал себя именно второй подход. Были разработаны такие инструменты как DataScript [1], FlexT[2], Kaitai Struct [3]. Их основным достоинством является то, что они позволяют сжато описывать спецификации форматов и автоматически генерировать устойчивые к ошибкам разборщики. В сфере анализа сетевых прокотолов были разработаны инструменты, адаптированные к особенностям сетевых пакетов: PacketTypes [4], BinPAC [5], GAPAL [6]. При этом никакого принципиального ограничения на использование их для описания именно сетевых протоколов нет.

В основе всех этих инструментов лежит универсальная модель формата данных. Эта модель должна обладать достаточной выразительностью для описания всех встречающихся форматов данных. Важно то, что наличие такой модели позволяет не только разбирать данные произвольного формата, но и проводить совместный анализ кода и данных. К этому ряду задач можно отнести восстановление форматов, отслеживание форматированных данных при исполнении программы.

Помимо разбора и анализа данных, актуальной задачей является задача генерации данных заданного формата для фаззинга систем. Использование единой модели данных на всем цикле восстановление-генерация-разбор сильно упрощает работу исследователю.

Таким образом, ключевой задачей при работе с форматами является разработка модели формата. Помимо требований функциональной выразительности, к ней должны предъявляться требования, учитывающие особенности возникающих задач. Так, например, в некоторых задачах разбора возникает необходимость разбирать неполные данные. В задачах восстановления форматов актуальна возможность модифицировать представление формата. В системах обнаружения вторжений (IDS), работающих сразу с базой форматов, возникает необходимость автоматически распознавать формат по данным.

В вышеуказанных инструментах основной упор делается на функциональную выразительность и безопасность, при этом мало внимания уделяется универсальности модели. В данной работе представлена разработанная модель формата данных, отвечающая сформулированным требованиям, реализованная на базе системы анализа бинарного кода Glassfrog. В разд. 2 приводится описание конструкций, часто встречающихся в форматах данных, которые модель должна описывать. Разд. 3 посвящен обзору существующих подходов и доступных программных инструментов для работы со структурированными данными. В разд. 4 предлагается универсальная модель формата данных, описываются декларативный язык для спецификации форматов и алгоритм разбора данных на основе предложенной модели. Разд. 5 и 6 содержат описание тестирования разработанной системы и направлений дальнейшей работы. В заключении формулируются отличительные черты предложенной модели и предполагаемые практические приложения.

2. Форматы данных

Среди распространенных общедоступных форматов можно выделить несколько групп:

- форматы сетевых протоколов: TCP, DNS, IPv4, ICMP;
- форматы исполняемых файлов: ELF, DOS MZ, Microsoft PE;
- мультимедийные форматы: GIF, PNG, PSD;
- и т. д.

Ниже приведены часто встречающиеся в описаниях форматов конструкции. Источниками описаний являлись официальные спецификации (в т.ч. соответствующие документы RFC).

- **Переиспользование форматов.** Поля формата сами по себе имеют некоторую структуру. Некоторые поля формата могут иметь одну и ту же структуру. В таком случае естественным решением будет описать эту структуру отдельно и указать ее как формат всех этих полей.
- **Параметризация формата.** Некоторые форматы имеют версии, отличающиеся только размером некоторого поля. В таких случаях целесообразно воспринимать этот формат как универсальный, зависящий от параметра – размера поля. Аналогично форматы могут отличаться лишь порядком байтов в многобайтовых последовательностях. В таком случае также удобно использовать порядок байтов как параметр. Другим примером необходимости параметризации является инкапсуляция сетевых протоколов – во многих протоколах данные полезной нагрузки также имеют некоторый формат. Этот инкапсулированный формат можно рассматривать как параметр верхнего формата и таким образом реализовать инкапсуляцию. Например, описав отдельно форматы сообщений протоколов TCP и UDP, мы можем передавать их в качестве параметра в протокол IP и получать связки TCP/IP и UDP/IP.
- **Условная структура.** Простейший случай – это опциональные поля, которые могут присутствовать, а могут отсутствовать в зависимости от некого условия (значения другого поля). Например, в формате сжатого файла GZIP может храниться имя исходного файла или не храниться, в зависимости от значения флага FNAME в заголовке. В более сложном случае вся структура полей формата может зависеть от некоторого условия. Например, формат одной секции (chunk) в файлах изображения PNG задается полем типа этой секции и может состоять из полей метаинформации об изображении, сжатого содержимого изображения, данных о палитре или быть вообще пустым.
- **Адресация полей: точка отсчета и отступ.** Некоторые поля в соответствии со спецификацией располагаются на определенном отступе от предыдущего поля или от некоторой другой точки отсчета (например, начала сообщения), причем величина отступа может определяться динамически. Например, в формате исполняемого файла

ELF содержатся таблицы заголовков программы и секций, отступ которых задается соответствующими полями в заголовке ELF.

- **Последовательности полей.** Часто встречаются форматы, в которых поля одной структуры повторяются некоторое количество раз. Это количество может быть фиксированным или определяться значениями других полей. Кроме того, последовательность полей может быть терминированной. Например, в формате сообщения протокола HTTP при использовании опции «Transfer-Encoding: chunked» структуры Chunk («куски») повторяются до тех пор, пока не встретится т.н. «нулевой кусок» (Chunk с телом нулевой длины). Другой пример: в формате сообщения протокола DNS поле заголовка QDCOUNT задает количество структур запросов в секции запросов.
- **Битовые поля.** В некоторых форматах размеры полей не выровнены по границам байтов. Типичным примером являются флаги – поля, состоящие из одного бита.
- **Потоковые данные.** Некоторые поля содержат сжатые или зашифрованные данные. В таком случае значение слова декодируется как битовый поток и может иметь переменную длину, определяемую динамически.
- **Валидация значений.** Поля форматов могут иметь фиксированное значение, определенное его спецификацией (т.н. magic-поля). В ряде случаев в спецификации указывается, что в случае несоответствия этому значению дальнейший разбор проводить не следует. Другим примером валидации значений являются поля контрольных сумм, используемые, например, в сетевых протоколах TCP и UDP.
- **Управление порядком байтов.** В некоторых форматах порядок байтов фиксируется в спецификации, однако есть форматы, которые хранят в себе данные переменного порядка байтов, причем определяется он динамически. Примером является формат исполняемого файла ELF.

Надо отметить, что в данной работе помимо достаточно сложных форматов (ELF, DNS, и т.п.) рассматриваются и более простые. Так, такие виды структурированных данных как структуры языка Си, списки, целые числа, также рассматриваются как форматы. Часто они используются как базовые форматы для построения более сложных форматов. В контексте задач анализа они также являются полезными для исследователя, поскольку позволяют типизировать тривиальные данные.

3. Подходы к описанию форматов

3.1 Синтаксический анализ

Понятие формата данных тесно связано с понятием формального языка. Классическая теория формальных языков описывается, например, в [7]. Формальный язык определяется следующим образом: пусть дано некое непустое конечное множество, называемое алфавитом, тогда словом называется произвольная (возможно, пустая) конечная последовательность элементов алфавита (символов), а языком – произвольное множество слов. Формат данных мы можем представить, как формальный язык. Например, текстовый формат сетевого протокола HTTP определяет способ записи сообщения в виде конечной последовательности символов ASCII, что, соответственно, задает некоторое множество слов в алфавите ASCII, т.е. некий язык. Аналогично, бинарный формат данных, например, формат исполняемого файла ELF, хранящийся в памяти, можно представить, как язык над алфавитом байтов.

Задача задания формального языка (в нашем контексте – описание формата) является базовой в теории формальных языков и имеет набор стандартных решений. Одним из базовых представлений языка являются формальные грамматики, классифицируемые с помощью иерархии Хомского по степени жесткости условий. Наиболее ограниченным типом

грамматик задаются регулярные языки, которые описываются регулярными выражениями. В рамках разработки языков программирования для синтаксического анализа наибольшее распространение получили контекстно-свободные грамматики (КС-грамматики).

Синтаксический анализ восстанавливает структуру разбираемого текста в соответствии с заданным языком. Этот вид анализа достаточно близок к задаче разбора данных в соответствии с заданным форматом. В теории формальных языков описываются алгоритмы построения машины разбора (анализатора) по КС-грамматике: LR-, LL(k)-, LALR-анализаторы и т.д., которые имеют доказательную базу своих свойств и хорошо показали себя за многолетнюю практику использования. Для записи КС-грамматик в компьютерной сфере часто используют форму Бэкуса-Наура (БНФ) или ее расширенную модификацию РБНФ.

Существуют инструменты, генерирующие разборщики по подобному представлению грамматики:

- yacc – программа для генерации разборщика по грамматике, описанной в форме БНФ с небольшими модификациями (вставками на языке Си), генерирует LALR-анализатор в виде кода на языке Си;
- Bison – GNU-версия yacc, которая может генерировать также LR-, IELR(1)- и GLR-анализаторы;
- ANTLR – использует для описания грамматики форму РБНФ и генерирует LL(*)-анализатор на различных выходных языках (Java, C#, C++, JavaScript, Python, Swift, Go).

Bison и yacc требуют для своей работы предварительной токенизации входных данных, т.е. лексический анализ (используются инструменты Lex или Flex).

3.2 Грамматики с зависимостями по данным

Синтаксический анализ имеет принципиальную особенность, не позволяющую использовать его для разбора форматов – независимость синтаксической структуры от *значений* ее элементов. Когда речь идет о произвольных форматах данных, структура результата разбора может меняться в зависимости именно от значений своих элементов. В таком случае поле разбираемой структуры интерпретируется, например, как число или строка, и используется в некотором выражении, определяющем структуру: условии, длине и т.п. Такую задачу называют «разбор с зависимостями по данным» (data-dependent parsing) и именно она соответствует задаче разбора форматов файлов и сетевых пакетов. Примером может являться распространенный шаблон формата: «поле длины + тело», где в начале задается фиксированного размера слово, интерпретируемое затем, как число, определяющее количество символов в «теле». Подобные форматы соответствуют формальным языкам, однако их описание с помощью классических контекстно-свободных грамматик затруднительно.

Формальные грамматики получили огромное функциональное развитие: атрибутные грамматики [8], грамматики, разбирающие выражения (PEG) [9] и т.д., что существенно повысило их выразительную мощность. Генерируемые на их основе разборщики являются модификациями вышеупомянутых «канонических» синтаксических анализаторов. Наиболее близкими к задаче разбора бинарных форматов можно считать т.н. грамматики с зависимостями по данным (data-dependent grammars, ЗД-грамматки). ЗД-грамматики являются расширением КС-грамматик и дополняют их привязанными семантическими значениями, которые могут быть использованы для наложения условий и вычисления выражений в процессе разбора. На их основе были разработаны такие инструменты для разбора форматов по описанию, как Yakker [10] и Iguana [11].

В работе [10] в качестве модели разборщика ЗД-грамматик описывается автомат с зависимостями по данным (data-dependent automaton), а также формально доказывается, что его мощности достаточно для разбора языков, задаваемых ЗД-грамматиками. Предлагаемый

авторами алгоритм разбора является модификацией хорошо известного алгоритма Эрли [12]. При этом авторы ставят перед собой задачу максимально эффективно использовать уже существующие алгоритмы разбора и их реализации, поскольку они за долгое время использования были хорошо оптимизированы. В связи с этим для спецификации формата было предложено использовать дополнительный язык промежуточного уровня, максимально близкий к языку описания контекстно-свободных грамматик [13].

Несмотря на достаточную выразительность моделей для описания практически произвольных форматов данных, в приведенных работах основной упор делается все ещё на разбор языков программирования. Подход с использованием модификаций грамматик все ещё страдает от недостаточного удобства для описания именно бинарных форматов: отсутствует возможность лаконично описывать такие опции, как порядок байтов и битовые поля. Тем не менее очевидным достоинством такого подхода является строгая доказательная база и высокая эффективность разборщиков, что является важной особенностью, когда речь идет о разборе языков программирования общего назначения.

В работе [14], однако, высказывается мнение, что подход с использованием КС-грамматик может быть эффективен в случае построения *валидатора* входных данных, который часто является частью разборщика. Авторы изучили конструкции сетевых протоколов с формальной точки зрения и предложили систему языков с доказательной базой эффективности их разбора. В частности, такие конструкции, как поля длины, проверки равенства, неравенства и выполнения численного сравнения, были описаны ими в формальном стиле.

3.3 Предметно-ориентированные языки

Альтернативный подход к описанию форматов данных заключается в разработке собственной модели формата и предметно-ориентированного языка, который будет транслироваться в нее. Такой подход позволяет учесть все требования к выразительности описания, сформулированные выше, и поэтому является более предпочтительным. При этом синтаксис и семантика такого языка будут закономерно сложнее тех, что использовались при описании грамматик.

Синтаксический вид языка может выбираться разработчиком исходя из соображений максимального удобства описания спецификаций. В таком случае разработчики часто останавливаются на конструкциях, сходных с описанием структур Си. На рис. 1 приводится пример описания формата ELF на языке спецификации форматов DataScript [1].

Другие авторы отталкивались от удобства разработки и избегали использования собственного синтаксиса. В таком случае за основу часто брался язык разметки (XML, YAML и т.п.) с определенной схемой, и разбор синтаксиса перекладывался на стандартный инструмент разбора данного языка. Так, например, в инструменте Kaitai Struct [3] для описания форматов используется язык YAML (рис. 2), а в системе фаззинга Peach Fuzzer [15] – язык XML.

Альтернативой использования декларативного языка описания спецификаций является использование API для конструирования представления формата. Такой подход используется, например, в инструменте обратной разработки и фаззинга Netzob [16] для описания форматов сообщений. Их декларативный язык ZDL является открытым API библиотеки Netzob, а файлы спецификаций представляют собой файлы кода на языке Python (рис. 3).

```
ElfSection(Elf32_File.Elf32_SectionHeader h) {
  Elf32_File::h.sh_offset:
    union {
        { } null : h.sh_type == SHT_NULL;
        StringTable(h) strtab : h.sh_type == SHT_STRTAB;
        SymbolTable(h) symtab : h.sh_type == SHT_SYMTAB;
        SymbolTable(h) dynsym : h.sh_type == SHT_DYNSYM;
        RelocationTable(h) rel : h.sh_type == SHT_REL;
        ...
    } section;
};

SymbolTable(Elf32_File.Elf32_SectionHeader h) {
    Elf32_Sym {
        uint32    st_name;
        uint32    st_value;
        uint32    st_size;
        uint8     st_info;
        uint8     st_other;
        uint16    st_shndx;
    } entry[h.sh_size / sizeof Elf32_Sym];
};
```

Рис. 1. Фрагмент описания формата ELF на языке DataScript
Fig. 1. ELF format specification in DataScript

```
meta:
  id: tcp_segment
  endian: be
seq:
  - id: src_port
    type: u2
  - id: dst_port
    type: u2
  - id: seq_num
    type: u4
  - id: ack_num
    type: u4
```

Рис. 2. Фрагмент описания сообщения протокола TCP на языке Kaitai Struct
Fig. 2. TCP message specification in Kaitai Struct

```
from netzob.all import *
f1 = Field(String(), name="field1")
f2 = Field(Integer(interval=(10, 100)), name="field2")
f3 = Field(Raw(nbBytes=14), name="field3")
symbol = Symbol([f1, f2, f3], name="symbol_name")
```

Рис. 3. Фрагмент описания на языке Netzob Description Language (ZDL)
Fig. 3. Modeling in Netzob Description Language (ZDL)

Такой императивный подход упрощает реализацию инструмента, поскольку авторам не требуется разрабатывать транслятор, однако синтаксические структуры получаются достаточно объемными, что увеличивает размер спецификации.

В синтаксическом анализе математическими объектами, на которых строится спецификация формата, являются *грамматика* и *формальный язык*. В основе моделей данного подхода часто лежат такие математические объекты, как *ориентированный граф* и *дерево*.

Вершинами первого обычно являются форматы данных, а ребра графа отражают появление указанного формата в описании полей исходного. Для корректной спецификации формата при этом необходимо определить расположение полей, например, пронумеровав ребра. Древовидные структуры часто возникают тогда, когда используются встроенные в модель форматы, описание которых не требуется. На рис. 4 приводится схематичное изображение модели в DataScript. На основе данной модели строится конкретный экземпляр спецификации, корневым элементом которого является глобальный *StructType*.

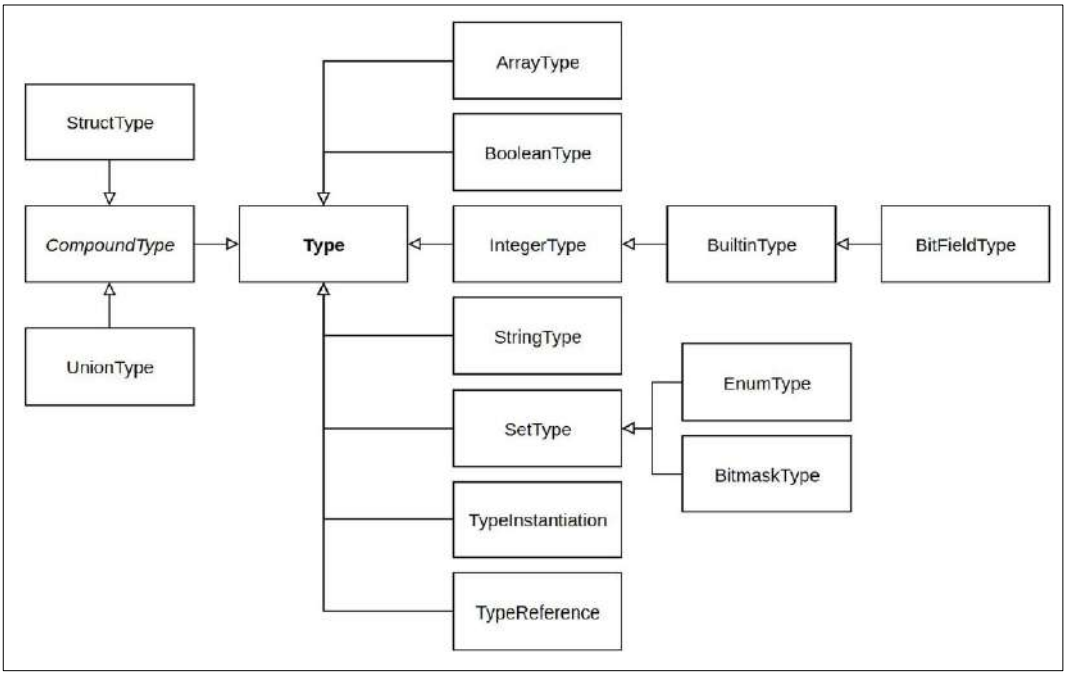


Рис. 4. Схематичное изображение модели в DataScript
Fig. 4. Types model schema in DataScript

Древовидные структуры в моделях позволяют описывать форматы с контекстом. Иными словами, появляется возможность использовать поля вышеопределенного формата. В случае отсутствия контекста, в модели может быть предусмотрено использование параметров формата для аналогичных целей.

Листовыми вершинами в графах форматов часто являются встроенные форматы. Наиболее распространенные встроенные форматы отражают основные типы переменных в языках программирования:

- байтовый массив;
- целые числа различных размеров;
- булево значение;
- битовый массив;
- строка (ASCII или Unicode).

Модели, нацеленные на описания специфичных форматов данных, например, сетевых протоколов, могут также иметь специфические встроенные форматы, такие как IP-адреса, битовые маски, строковые разделители (CRLF). Для построения пользовательского формата из встроенных форматов используются агрегирующие форматы, такие как структура или массив.

Помимо этого, модели часто реализуют специальные директивы для описания сжатых полей (например, с помощью алгоритма GZIP). Такие директивы часто реализованы вручную заранее описанным алгоритмом декодирования. Некоторые модели относят кодирование строк к этому же случаю и позволяют описывать строки различных кодировок.

3.4 Существующие инструменты

Инструменты для описания форматов данных можно разделить по сферам применения. Исторически одной из первых постановок задачи декларативного описания формата была постановка в контексте описания сообщений сетевых протоколов, появившаяся в период активного развития сети Интернет. Тем не менее многие из этих инструментов не имеют принципиальных ограничений на использование для описания произвольных форматов данных. Авторы работы [4] представили свой инструмент PacketTypes как язык спецификации сетевых пакетов, способный описывать конструкции, встречающиеся в сетевых протоколах: например, инкапсуляция протоколов, поля переменного размера и опциональные поля. Авторы в том числе стремились с помощью автоматической генерации разборщиков решить проблему безопасности кода, работающего с сетями.

В дальнейшем инструменты для описания сообщений сетевых протоколов часто интегрировались в различные системы обнаружения вторжений (NIDS): например, инструмент BinPAC [5] был интегрирован в IDS Bro. Авторы BinPAC также уделяли большое внимание безопасности кода разборщика. Основной группой форматов, на которую они ориентировались, были сетевые протоколы уровня приложения. В их работе также отмечается, что для описания сообщений сетевых протоколов инструменты, основанные на грамматиках, такие как yacc и ANTLR недостаточно выразительны. Той же группой авторов в дальнейшем был разработан аналогичный инструмент Spicy [16], который также интегрирован в IDS Zeek (последователь Bro), однако позиционируется как генератор разборщиков произвольных форматов.

При разработке инструментов описания сетевых протоколов перед разработчиками встают задачи описания не только форматов сообщений, но также автоматов состояний протоколов и сессий. В связи с этим в подобных инструментах часто для разбора данных и поддержания состояний других элементов используется *внутренний движок*. В работе [6] описывается система GAPA (Generic Application-Level Protocol Analyzer) и используемый в ней язык GAPAL. GAPA ориентирован на анализ всего сетевого трафика на всех слоях. В GAPAL основным примитивом для описания является *протокол*, включающий в себя описания нижележащих протоколов, грамматики сообщений, автомата состояний и обработчиков сообщений. Для описания форматов сообщений GAPA использует комбинирование подхода описания Си-подобными структурами и КС-грамматиками. Как указывают авторы, их главной задачей было поддерживать описания бинарных и текстовых протоколов внутри одного представления.

Параллельно с инструментами анализа сетевого трафика та же задача декларативного описания формата ставилась и для произвольных форматов. В работе [19] описывается декларативный язык DataScript. Функциональной выразительности языка достаточно для описания очень широкого класса форматов. Однако гибкости в способе задания отступов не всегда хватает для описания пересекающихся или произвольно расположенных полей. Как отмечается автором, декларативное описание формата может быть использовано не только для разбора данных, но и для генерации данных описанного формата.

Как и в сфере сетевых протоколов, к инструментам разбора произвольных форматов предъявлялись высокие требования безопасности разборщиков. Так, в работе [18] авторами предлагается математический аппарат – исчисление описаний форматов (data description calculus, DDC) – основанный на теории типов, и на его основе формулируется ряд утверждений касательно разборщиков и возникающих в них ошибках. На основе этого аппарата авторы проанализировали языки инструментов PacketTypes и DataScript. В инструменте PADS [19], разработанном авторами вышеуказанного аппарата, особое внимание уделяется механизмам обработки ошибок в разборщиках.

В системах фаззинга выделяют два подхода к получению данных: мутация и генерация. Первый подход подразумевает внесение случайных изменений в существующие входные данные. Во втором подходе возникает необходимость описывать форматы данных. Подсистема фаззера принимает на вход описание формата и генерирует данные в соответствии с предоставленной спецификацией. В связи со спецификой использования представления формата, в инструментах обычно не реализована возможность разбирать данные. Часто встречающейся разновидностью фаззинга является фаззинг сетевых протоколов, по этой причине в данных инструментах для описания форматов можно встретить примитивы, характерные для сетевых протоколов (например, примитивы протокола HTTP).

Отличительной особенностью таких моделей является повышенное внимание к валидируемым полям: т.н. magic-полям, контрольным суммам и т.п. В моделях встречаются встроенная поддержка ограничений на значения полей и предикаты корректности.

Многие из данных инструментов (BooFuzz [20], Netzob [16]) не используют декларативный язык для описания форматов, а вместо этого позволяют создавать экземпляры форматов с помощью API. Однако один из наиболее известных инструментов фаззинга Peach Fuzzer [15] (в 2020 году был выкуплен компанией GitLab и распространяется под названием GitLab Protocol Fuzzer [21]) получает на вход описание модели данных в виде XML-файла определенной схемы.

Среди инструментов, способных обрабатывать форматированные данные, отмечаются также редакторы бинарных файлов. Изначально поддержка форматов обеспечивалась в них по первому сценарию разработки разборщика, т.е. реализовывалась вручную для каждого необходимого формата. Описание пользовательских форматов было весьма ограничено и сводилось к простым структурированным полям фиксированной длины. На текущий момент были разработаны такие инструменты как 010 Editor [22] и GNU poke [23], которые позволяют описывать пользовательские форматы произвольной сложности. Такие инструменты имеют графический интерфейс, что существенно повышает наглядность и позволяет визуализировать процесс разбора данных.

Для изучения свойств представления формата необходим доступ к исходному коду инструмента. Некоторые из упомянутых инструментов (GAPA, 010 Editor) не имеют открытого кода, поэтому их изучение затруднено.

В табл. 1 представлена краткая сводка по рассмотренным инструментам, способным разбирать бинарные форматы. Среди критериев сравнения выделяются в первую очередь интерфейсы взаимодействия с инструментом: входной язык описания формата и интерфейс разбора данных. Из таблицы можно заключить, что большинство инструментов используют язык с собственным синтаксисом. Многие из этих языков имеют внешнее сходство в синтаксисе со структурами в языке Си (DataScript, BinPAC, PADS, Nail), другие содержат синтаксические конструкции, схожие с правилами вывода в грамматиках (Spicy, GAPAL). Некоторые инструменты используют для декларативного описания синтаксис языков представления данных YAML или XML (NetPDL в системе NetBee, Kaitai Struct).

Среди изученных инструментов только Netzob и BooFuzz представляет возможность манипулировать представлением формата с помощью API, который при этом является и

единственным способом спецификации формата. Таким образом в большинстве случаев разработчики не предоставляют возможности создавать экземпляры представлений программным способом.

Табл. 1. Существующие инструменты работы с форматами данных
Table 1. Existing data formats tools

	Спецификация формата	**Разборщик**	**Модель**	**API модели**	**Открытый код**	**Поддержка проекта**
Kaitai Struct	YAML-based	кодогенерация (C++, Java, Python, ...)	дерево	×	✓	✓
FlexT	DSL[1]	внутренний интерпретатор	?	×	×	✓
DataScript	DSL	кодогенерация (Java)	орграф с корнем	×	✓	×
PADS	DSL	кодогенерация (C, OCaml)	–	×	✓	×
Spicy	DSL	кодогенерация (C++)	орграф	×	✓	✓
BinPAC	DSL	кодогенерация (C)	–	×	✓	×
NetBee	XML-based	внутренний	–	×	✓	×
GAPAL	DSL	внутренний	?	?	×	?
PacketTypes	DSL	кодогенерация (C)	?	?	×	?
GitLab Protocol Fuzzer (ex-Peach)	Peach Pit (XML)	–	орграф	×	*Community Edition*	✓
BooFuzz	API	–	–	✓	✓	✓
Netzob	API	–	–	✓	✓	×
FormatFuzzer	DSL (010 Editor)	встроенный	дерево	×	✓	✓
010 Editor	DSL	встроенный	?	×	×	✓
GNU poke	DSL	встроенный	–	×	✓	✓

Другим критерием сравнения является способ дальнейшего взаимодействия с представлением формата. Можно условно выделить два сценария работы данных инструментов: внутренний и внешний. Внутренний сценарий подразумевает трансляцию описания формата в промежуточное представление (модель формата) и дальнейшее использование объекта этой модели модулями системы (например, для разбора данных). Внешний же сценарий может быть рассмотрен как дополнение к внутреннему: в нем на основе описания формата инструментом генерируется код разборщика формата на одном из поддерживаемых языков программирования. Этот процесс кодогенерации может быть

[1]DSL – domain-specific language (предметно-ориентированный язык)

рассмотрен как следующая фаза после трансляции описания формата в модель, т.е. внешний сценарий может дополнять внутренний. В табл. 1 в соответствующем столбце в скобках указываются целевые языки, на которых инструмент имеет возможность генерировать разборщики.

3.5 Выводы

Основываясь на изучении существующих инструментов и подходов к решению задачи декларативного описания формата данных, можно однозначно заключить, что не было найдено достаточно универсальной модели для покрытия заданного набора задач, связанных с форматированными данными. Многие инструменты качественно решают задачу генерации универсальных разборщиков, но не предоставляют возможности использовать модели формата для задач совместного анализа кода и данных. Несмотря на большую выразительность многих декларативных языков, большинство из них неявно используют предположение об определенном способе хранения данных (сетевой поток, файл и т.п.), что затрудняет реализацию таких опций, как разбор с частично недоступными данными.

4. Предлагаемое решение

Разрабатываемая в ИСП РАН система анализа бинарного кода *Glassfrog* [24] покрывает задачи совместного анализа кода и данных и поэтому нуждается в спецификации форматов данных. Ключевым элементом в системе *Glassfrog* является платформонезависимое представление кода *Pivot* [25]. Предлагаемая модель формата данных была реализована как часть этого представления и в связи с этим имеет доступ к другим его сущностям.

Система *Glassfrog* представляет возможность описывать архитектуры наборов команд (ISA), их синтаксис и семантику. Для этого разработан предметно-ориентированный язык *Ribbit*, который транслируется во внутреннее представление *Glassfrog* (*Pivot*-модуль); этот же язык используется для спецификации форматов данных. На рис. 5 представлен стандартный тракт обработки форматированных данных с помощью системы *Glassfrog*. Ниже в работе описываются разработанная модель формата и алгоритм разбора данных по ней, приводятся примеры спецификации форматов на языке *Ribbit*.

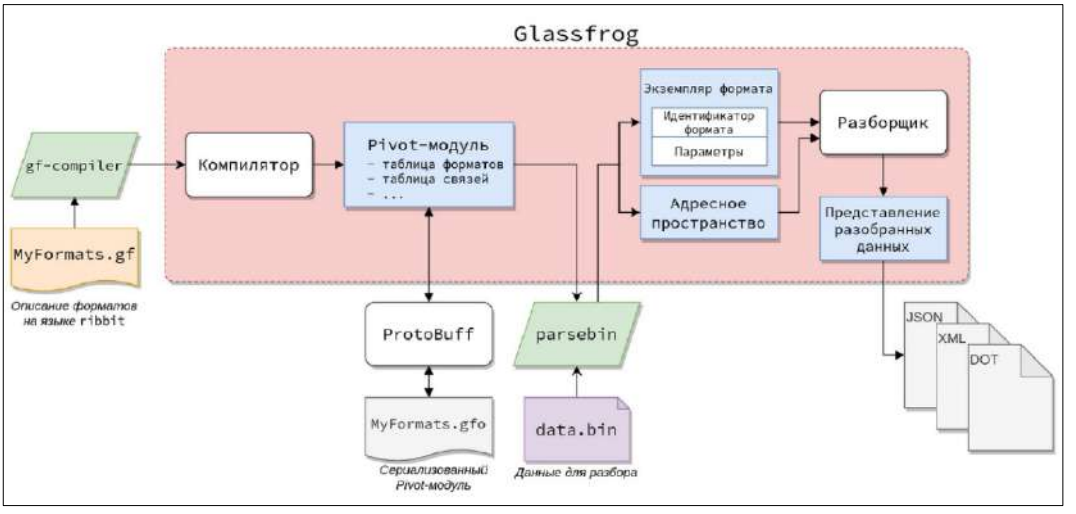


Рис. 5. Тракт разбора данных в системе Glassfrog
Fig. 5. Parse pipeline in Glassfrog

4.1 Модель формата

Ключевым элементом данной модели является формат (*DataFormat*). Каждый формат имеет набор параметров (*DataArg*), значения которых могут быть использованы при определении его структуры. Задав значения всех параметров определенного формата, мы таким образом конкретизируем его. Формат с выбранными значениями его параметров называется *экземпляром формата*. Сам по себе формат не является завершенной сущностью, разбор данных можно проводить только по экземпляру формата.

Основным отличием предлагаемого подхода от большинства изученных является отхождение от концепции полей как последовательных отрезков байтов в буфере. В модели предлагается трактовать поля как связи (*Relation*) между различными форматами. Каждый формат имеет набор (вообще говоря, неупорядоченных) связей. Все связи разделяются на три типа (рис. 6):

- внутренняя связь (*Internal*);
- внешняя связь (*External*);
- связь-значение (*Value*).

Внутренние связи описывают концепцию обыкновенных полей формата. В самом простом случае они состоят из экземпляра формата и структуры, описывающей ее положение. На этапе разбора данных для каждой внутренней связи рекурсивно разбирается ее формат данных.

Поскольку от модели требуется описание условных конструкций в структуре формата, указанный экземпляр формата и положение могут обособляться предикатами, определяющими выбор одного формата (или положения) из нескольких возможных. Итоговая структура представляет собой конструкцию, напоминающую switch-case: в представлении хранится массив из пар «предикат + вариант реализации». Вычисление предикатов осуществляется по порядку. Результатом вычисления такой конструкции будет тот вариант реализации, который соответствует первому найденному истинному предикату. Сам предикат представляет собой функцию, аргументами которой могут являться значения аргументов формата или значения других связей (внутренних или связей-значений).

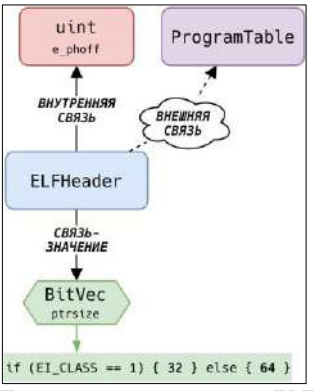


Рис. 6. Пример связей в заголовке ELF
Fig. 6. ELF header relations example

Внешние связи по своему строению не отличаются от внутренних: они также состоят из экземпляра формата и положения. Однако в отличие от внутренних, на этапе разбора данных форматы внешних связей не разбираются рекурсивно. В результате разбора будет храниться только структура экземпляра формата вместе с вычисленным положением. Это позволяет пользователю контролировать, какие связи он хочет разбирать, а какие хочет лишь указать. Также это позволяет разбирать неполные данные: разобрав имеющуюся часть, пользователь

отмечает поля из недоступной области как внешние и тем самым сохраняет информацию об их форматах, не разбирая их.

Связи-значения отличаются от предыдущих по своему строению. Они не имеют положения и представляют собой вычисляемое выражение некоторого типа. Значения этих связей могут быть использованы при определении формата или положения других связей. На этапе разбора данных эти значения вычисляются и сохраняются в результате разбора. Помимо удобства пользователя при определении других связей (сокращения количества кода), это позволяет описывать поля, состоящие из произвольного числа битов.

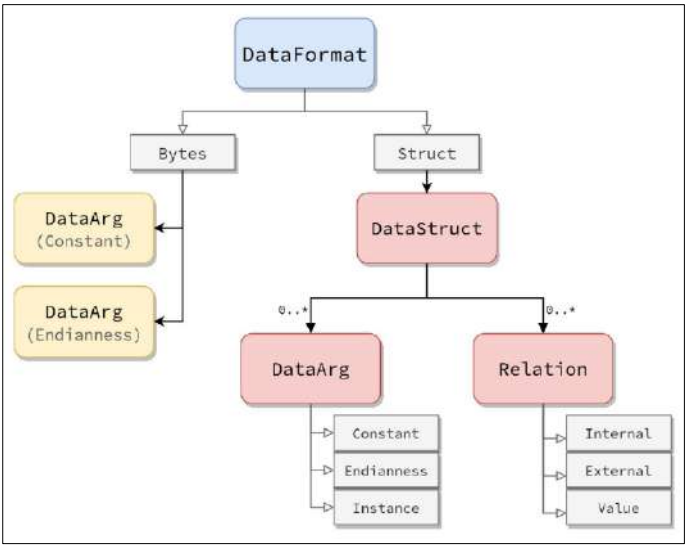


Рис. 7. Модель формата данных
Fig. 7. Data format model

В модели используется единственный встроенный формат *Bytes* – формат байтов определенного размера с определенным порядком байтов (little-endian/big-endian), используемым при чтении. Этот формат не имеет связей и по сути является самым примитивным способом структурировать данные. Все остальные форматы являются составными и определяются пользователем (*DataStruct*). На рис. 7 схематично иллюстрируются встроенный и пользовательские форматы.

Важным элементом модели является структура, описывающая положения внутренних и внешних связей (*Location*). Так же, как и формат связи, ее положение может обособляться предикатами, позволяющими описывать его как условное выражение. В результате поле формата может иметь динамически определяемое положение. Сама структура состоит из трех частей (рис. 7).

- **Объект отсчета** (*Origin*) – определяет, к какому объекту будет привязана связь. Возможны три способа привязки: к другой связи (*Relation*), к описываемой структуре (*Data*) и ко всему адресному пространству (*Const*). Привязка к другой связи определяется при помощи ее идентификатора (*RelationId*).
- **Якорь** (*Anchor*) – объект отсчета определяет связь, структуру или пространство, то есть во всех случаях некоторый диапазон адресов, от начала и до конца объекта. Якорь указывает, что именно необходимо выбрать: начало или конец объекта.
- **Отступ** (*Offset*) – задает отступ в байтах от определенной двумя предыдущими частями точки.

Стандартным положением связи считается конец предыдущей описанной связи с нулевым отступом. Такие положения автоматически выводятся компилятором *Glassfrog* и пользователю не придётся описывать их для каждого поля.

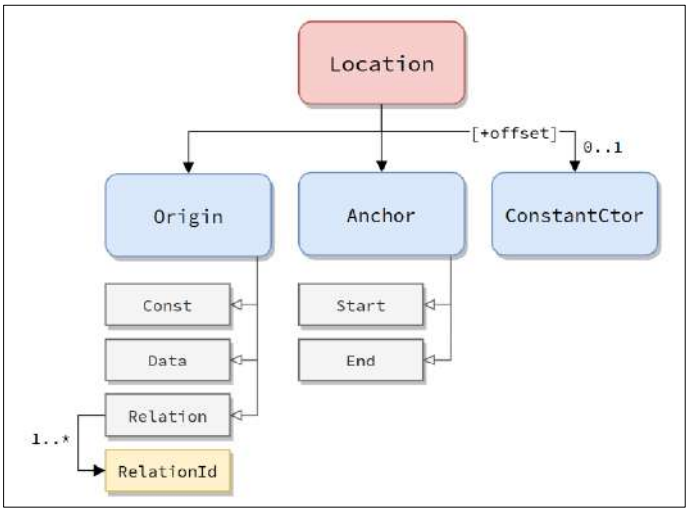


Рис. 8. Положение связи
Fig. 8. Relation location

Для представления форматированных данных используется структура *Data* и вспомогательная структура *DataPtr* (рис. 8). Структура *Data* хранит только разметку и не хранит сами данные внутренних связей. Для каждого вида связи хранятся соответствующие объекты:

- для *Internal* хранится объект *Data*;
- для *External* хранится объект *DataPtr*;
- для *Value* хранится значение одного из трех типов: битовый вектор, порядок байтов (*endianness*) или экземпляр формата.

Здесь видно, что для внешних связей не хранится внутреннее устройство их формата, а только его параметры и адрес, где находится связь.

На данном этапе реализации значения связей (внутренних и связей-значений) не валидируется. В дальнейшей работе планируется добавить обособление связей предикатами, вычисляемыми в процессе разбора и накладывающими ограничения на значения этих связей (аналог *assert*). В случае ложности данного предиката алгоритм разбора будет сразу завершаться, возвращая соответствующее исключение. Это позволит корректно обрабатывать такие примитивы как поля с фиксированным значением (magic-поля), поскольку в спецификациях подобных форматов часто указывается, что дальнейший разбор в случае несоответствия magic-поля требуемому значению проводить не следует.

Отличительной особенностью данной модели является исключительная гибкость в задании положений полей. Это позволяет описывать такие сложные структуры как закодированное доменное имя в сообщениях протокола DNS, не прибегая к написанию дополнительного кода. Интегрированность модели в представление *Pivot* даёт потенциал для использования модели в задачах совместного анализа кода и данных.

4.2 Алгоритм разбора

Поскольку предложенная модель существенно отличается от представления грамматик, для разбора не мог быть адаптирован один из существующих алгоритмов, и описываемый алгоритм был разработан «с нуля». На текущем этапе к алгоритму не предъявлялось

требований относительно скорости работы, однако такие требования, как завершимость, были удовлетворены (гарантируется, что разборщик не войдет в вечный цикл). Псевдокод алгоритма приводится в приложении 1.

Основным принципом работы алгоритма является принцип вычислимости. Считается, что связь вычислима, если вычислимы ее предикаты, положение и экземпляр формата (для внутренних и внешних связей) или определяющее ее выражение (для связей-значений). Положение связи может зависеть от других связей и станет вычислимым только тогда, когда все они будут вычислены. Аналогично параметры формата могут зависеть от других связей и экземпляр станет вычислимым, когда все они будут вычислены. Важно отметить, что в случае если предикат связи вычислим и истинен, другие предикаты и привязанные к ним экземпляры форматов вычисляться не будут. Когда экземпляр формата внутренней связи вычислен, алгоритм рекурсивно начинает вычислять его структуру.

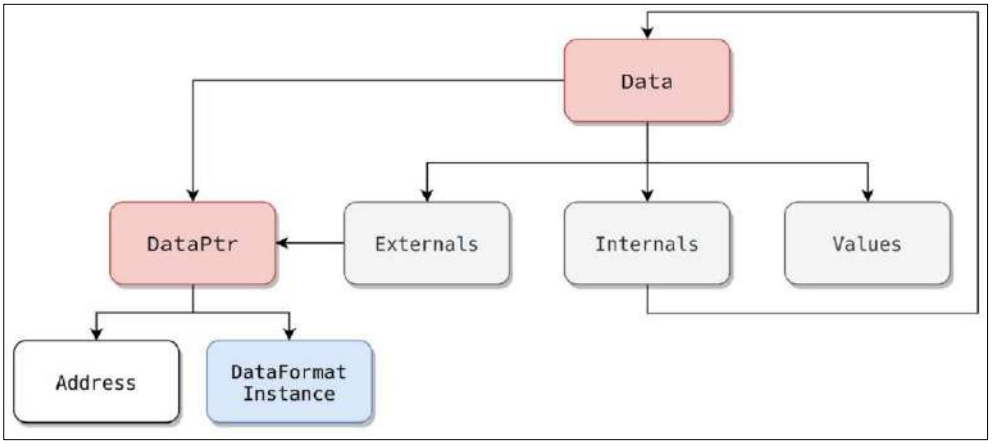


Рис. 9. Представление результата разбора
Fig. 9. Parse result representations

В процессе своей работы алгоритм определяет вычислимые на данном этапе связи и вычисляет их, сохраняя результат в структуру *Data* (рис. 9), и затем переходит на следующий этап. Если на данном этапе не было вычислено не одной связи, то в зависимости от того, остались ли в формате невычисленные связи или нет, алгоритм завершится с ошибкой или успешно.

Такой подход позволяет разбирать форматы с циклическими зависимостями связей, которые семантически корректны. Пример приведен на рис. 10, где изображен формат с двумя связями A и B, которые меняются положениями в зависимости от внешнего параметра P.

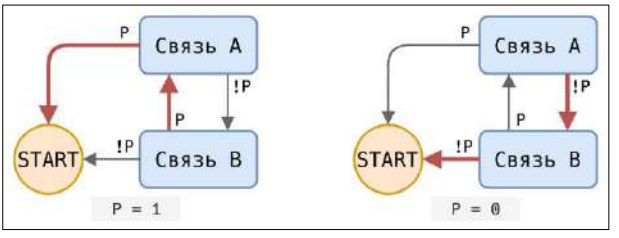


Рис. 10. Пример циклических зависимостей связей
Fig. 10. Relations cyclic dependencies example

Здесь их положения имеют циклическую зависимость на этапе компиляции, однако разрешаются корректно на этапе разбора. Реальным примером такого поведения является структура заголовка программы в формате исполняемого файла ELF (поля p_offset и p_flags).

Таким образом, порядок вычисления связей определяется динамически. Чтение самих данных из адресного пространства производится только в том случае, когда необходимо использовать это значение. В противном случае данные не читаются.

Алгоритм разбора обходит связи в порядке их указания в спецификации, поэтому в простейшем случае весь формат будет разобран за один проход. В худшем случае сложность работы алгоритма составляет $O(n^2)$, где n – количество связей формата. Оптимизация алгоритма, заключающаяся в построении графа зависимостей связей на этапе компиляции, позволяет достичь линейного времени работы алгоритма и запланирована в дальнейшей работе.

4.3 Язык Ribbit

Для спецификации форматов были реализованы необходимые синтаксические конструкции в языке *Ribbit*. Получившийся декларативный язык описания формата позволяет использовать всю выразительную мощность модели и одновременно является достаточно компактным за счет использования собственного синтаксиса.

В *Ribbit* в качестве численных переменных используются битовые вектора фиксированного размера:

```
let x: '16 = 0;
```

Битовые операции используют префиксную запись с апострофом в начале названия операции:

```
'add(1'32, 2'32).
```

```
data MyFormat(S: '64, E: endianness, F: data) {
    // Внутренняя связь
    magic: bytes(5, little),

    // Явное задание положения связи
    header: at(
        +(end(magic), 4),
        bytes(1, little)
    ) as '8,

    // Привязанное значение (битовый флаг)
    val flag: '1 = header[0],

    // Опциональное поле
    if flag {
        body: F,
    },

    // Связь с переменным форматом
    crc: if flag { CRC32() } else { CRC16() },

    // Внешняя связь
    extern tail: MyAnotherFormat(S, E),
}
```

```
      magic                    header → flag = 1   body
4d 59 46 4d 54 00 00 00 00 01 aa bb cc dd ee ff
3a 67 6c 1c ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ..
      crc          MyFormat          tail
```

Рис. 11. Наверху: пример описания формата на языке Ribbit, внизу: пример разобранного буфера
Fig. 11. Top: example of format specification in Ribbit, bottom: example of parsed buffer

На рис. 11 приводится пример описания искусственного формата MyFormat на языке *Ribbit*, демонстрирующий возможности языка. Для тех полей, для которых не указано явно положение, компилятор автоматически выведет его. Поле tail соответствует внешней связи, поэтому его данные в буфере и размер обозначены как неизвестные.

В языке нет встроенной поддержки массивов, вместо этого используется вспомогательный формат list, представляющий собой структуру, схожую с односвязным списком (рис. 12). Возможность описывать битовые поля реализована через *Value*-связи типа битового вектора (см. поле val на рис. 11).

Разработанный алгоритм разбора имеет ряд ограничений, необходимых для завершимости. Например, конструкцию полей «тело + длина» (именно в указанной последовательности) невозможно разобрать, поскольку в ней имеются явные кольцевые зависимости: размер поля «тело» зависит от поля «длина», а положение поля «длина» зависит от положения поля «тело». В некоторых случаях (например, в описанном выше) компилятор способен выявлять такие конструкции и сразу выдавать сообщение об ошибке в спецификации формата. При этом описанный формальный язык сам по себе существует и данное сообщение об ошибке следует интерпретировать как то, что алгоритм не способен разобрать данный формат и гарантированно войдет в вечный цикл.

```
data list(len: '64, T: data) {
    if 'ne(len, 0) {
        head: T,
        tail: list('dec(len), T),
    }
}
```

Рис. 12. Описание массива на языке Ribbit
Fig. 12. Array specification in Ribbit

5. Примеры использования

На разработанном языке были описаны форматы:

- сообщения сетевого протокола DNS;
- исполняемого файла ELF;
- файла с растровой графикой PNG.

Хотя задача разработки оптимального по скорости разборщика не ставилась, были проведены сравнительные тесты времени разбора пакетов DNS трафика. Результаты представлены в табл. 2. Разборщик *Glassfrog* проигрывает по скорости другим решениям, однако в абсолютном значении показывает удовлетворительный результат. Стоит учесть, что для полностью корректного сравнения скорости работы следует уточнить требуемое выходное представление данных и детализацию спецификации протокола (необходимые поля). При этом в разборщике имеется широкий диапазон возможных оптимизаций, которые могут существенно увеличить его производительность. Например, с учетом вида представления форматов и результатов разбора, некоторые структуры результатов могут быть построены уже на этапе компиляции, поскольку имеют фиксированные размеры и адреса.

Табл. 2. Сравнительные тесты производительности разборщиков
Table 2. Parser efficiency comparison

Glassfrog (C API)	Kaitai Struct (C++)	Spicy (C++)
1,586,230 packets	1,891,769 packets	1,585,275 packets
7,412 packets/s	196,224 packets/s	52,156 packets/s
0,7 Kbytes/s	18 Mbytes/s	3,94 Mbytes/s

Разработанный язык отличается в том числе своей компактностью. Результат сравнения размера описаний форматов на различных языках[2] приведен в табл. 3. Для сравнения взяты инструменты Kaitai Struct и Spicy, поскольку они имеют в открытом доступе обширные базы описанных форматов данных.

Табл. 3. Сравнение количества строк в описаниях форматов
Table 3. Line number comparison in data formats specifications

	Glassfrog	**Spicy**	**Kaitai Struct**
DNS	57	89	214
ELF	66	91	331
PNG	39	35	94

6. Направления дальнейшей работы

На данном этапе работы реализация содержит ряд ограничений, обход которых является первостепенной задачей в дальнейшей работе.

6.1 Ограничения предлагаемого решения

Одной из главных трудностей при описании форматов данных с использованием *Ribbit* является описание полей, не выровненных по границам байтов. В случае использования форматом алгоритма сжатия или иного битового кодирования возникают сложности при спецификации. Например, при описании формата PNG, некоторые поля которого кодируются с помощью алгоритма сжатия gzip, возникает проблема с тем, что не удается вычислить общий размер поля. При этом использование битовых полей в качестве флагов покрывается возможностями *Value*-связей и взятием части от разобранного поля. Возможным решением этой проблемы является расширение множества встроенных форматов форматом битов.

Другой проблемой, возникающей при описании формата, является необходимость валидировать значения полей переменного размера. Частным примером является использование контрольных сумм, которые вычисляются от всего массива данных, который не имеет заранее известного размера.

Язык *Ribbit* имеет возможность описания функций, оперирующих с битовыми векторами фиксированной длины. Требование фиксированности длины здесь критично, поскольку компилятору необходимо проверять корректность используемых типов. Для описания функций с различными типами используются шаблоны, которые явно раскрываются в конкретные реализации функций.

В нашем случае длина битового вектора, для которого необходимо вычислить контрольную сумму, определяется динамически. Таким образом, компилятор не может сгенерировать определенную функцию даже с использованием шаблонов. Потенциальным решением может быть описание функции, принимающей указатель на данные (адресные пространства и указатели также поддерживаются в *Ribbit*).

При описании формата с полем в виде списка сущностей способом, описанным выше, с использованием формата list, разобранные данные представляют собой многократно вложенную структуру (рис. 13).

Такое представление не слишком удобно для пользователя, желающего оперировать с полем, как с массивом. Возможным улучшением будет использование inline-директив, которые

[2] Поскольку в различных языках существуют различные примитивы, брались частичные описания формата, сходные по структуре.

позволят явно подставлять поля одного формата в другие. В таком случае возможно реализовать обращение к элементам списка по индексам.

```
Struct(T)
  _: Struct(list)
      head [0..2] ------------------------------------------- 0x457f
      tail: Struct(list)
          head [2..4] ----------------------------------------- 0x464c
          tail: Struct(list)
              head [4..6] ------------------------------------- 0x0102
              tail: Struct(list)
                  head [6..8] --------------------------------- 0x0001
                  tail: Struct(list)
                      head [8..10] ------------------------------ 0x0000
tail: Struct(list)
```

Рис. 13. Результат разбора формата списка list
Fig. 13. Parse result of list format

С целью упрощения спецификации форматов пользователем планируется разработка базы стандартных форматов, включающей в себя такие форматы как числа фиксированных размеров, списки, терминированные строки и т.п. Планируется также разработка инструмента для трансляции спецификаций распространенных форматов из других языков, обладающих обширными базами описанных форматов (например, из Kaitai Struct). С целью повышения производительности разборщика данных, планируется провести ряд оптимизаций реализации алгоритма: предварительное построение графа зависимостей связей, кэширование результатов вычислений выражений.

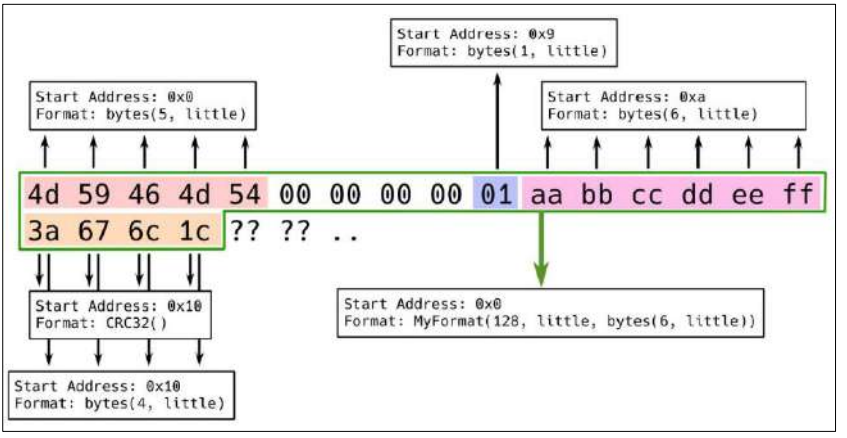


Рис. 14. Разметка буфера абстрактного состояния с использованием модели форматов
Fig. 14. Abstract buffer state markup using proposed data format model

6.2 Применение к задачам анализа бинарного кода

Одной из задач анализа кода, связанных с форматированными данными, является задача вывода типов данных. В процессе анализа бинарного кода частично восстанавливаются буферы памяти – определяются значения для некоторых адресов памяти. При дальнейшем изучении иногда удаётся определить, что конкретный буфер представляет собой не просто набор байтов, а вполне известную структуру, иными словами, имеет определенный формат. В таком случае информация об этом формате может быть полезна при дальнейшем анализе: анализируя последующие инструкции есть возможность отследить перемещение байтов

этого буфера, но теперь эти отслеживаемые байты будут представлять собой не просто помеченные данные, а данные определенного формата.

В простейшем случае, подобно анализу помеченных данных, можно отслеживать копируемые отрезки байтов и снабжать их разметкой формата. На рис. 14 приводится пример размеченного буфера с использованием модели форматов *Glassfrog*. Каждому биту сопоставляется экземпляр разобранных данных и смещение внутри него, соответствующее адресу бита. Разбор полей формата добавляет на те же самые биты новую разметку, использующую форматы полей. Эта разметка может использоваться как абстрактное состояние программы и с использованием абстрактной интерпретации [26] может быть решена задача вывода типов.

6.3 Применение к задачам генерации данных

Разработанная модель имеет потенциал для применения к задаче генерации данных. При генерации данных в соответствии со спецификацией может потребоваться генерировать как корректные данные, так и данные с ошибками (не соответствующие формату). Например, одним из видов ошибки может быть нарушение предиката корректности значения поля.

Для генерации корректных данных может быть модифицирован разработанный алгоритм разбора данных. В алгоритме чтение из пространства данных происходит только тогда, когда значение поля необходимо для вычисления некоторого выражения. В алгоритме генерации в этом случае будет необходимо сначала сгенерировать случайное значение, поместить его в пространство данных по требуемому адресу, а затем, также как в алгоритме разбора, прочитать его. Это позволит добиться корректности таких конструкций как «длина+тело». Поля, не использованные в выражениях, следует в конце заполнить произвольными данными. Такой подход не нарушит корректности структуры, однако может вызвать несоответствие предикатам корректности значений полей.

Для удовлетворения данным предикатам предполагается использование возможностей инфраструктуры Glassfrog, а именно символьного исполнения. Символьное исполнение позволит найти предикат на входные данные, удовлетворение которому будет обозначать истинность предиката корректности. Для него будет необходимо решить задачу выполнимости формулы (найти удовлетворяющие входные данные).

Символьное исполнение может также позволить изучать условные ветвления формата на предмет покрытия всех ветвей при генерации. Практические эксперименты с данными подходами запланированы в дальнейшей работе.

7. Заключение

В данной работе проводится анализ существующих подходов к декларативному описанию форматов данных и предлагается собственное решение, включающее модель формата и инфраструктуру для его спецификации. В ходе разработки были учтены выделенные ключевые особенности встречающихся форматов данных. Реализованный компилятор позволяет транслировать декларативное описание формата в представление, которое может быть использовано для задач разбора данных и совместного анализа кода и данных. Отличительными особенностями предлагаемого решения являются гибкость при задании положений полей формата, возможность конструирования и модификации представления с помощью API и, как следствие, возможность использовать модель для программного анализа форматированных данных. Разработанная инфраструктура была применена к спецификации распространенных форматов сообщений сетевых протоколов и исполняемых файлов.

Главными направлениями дальнейшей работы являются расширение модели для обработки битовых потоков и данных, не выровненных по границам байтов, и разработка механизмов вычисления функций от значений полей произвольного размера, в частности, для вычисления и валидации контрольных сумм.

Возможными прикладными приложениями модели к задачам анализа бинарного кода является использование представления при восстановлении форматов по трассе выполнения или образу программы, а также разметка данных памяти в задаче вывода типов.

Список литературы / References

[1] G. Back. DataScript - A specification and scripting language for binary data. Lecture Notes in Computer Science, vol. 2487, 2002, pp. 66-77.

[2] Хмельнов А.Е., Бычков И.В., Михайлов А.А. Декларативный язык FlexT - инструмент анализа и документирования бинарных форматов данных. Труды ИСП РАН, том 28, вып. 5, 2016 г., стр. 239-268 / Hmelnov A.Y., Bychkov I.V., Mikhailov A.A. A declarative language FlexT for analysis and documenting of binary data formats. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 5, 2016, pp. 239-268 (in Russian). DOI: 10.15514/ISPRAS-2016-28(5)-15.

[3] Kaitai Struct: declarative binary format parsing language. URL: https://kaitai.io/.

[4] McCann P.J., Chandra S. Packet Types: abstract specification of network protocol messages. ACM SIGCOMM Computer Communication Review, vol. 30, issue 4, 2000, pp. 321-333.

[5] Pang R., Paxson V. et al. binpac: a yacc for writing application protocol parsers. In Proc. of the 6th ACM SIGCOMM Conference on Internet Measurement (IMC '06), 2006, pp. 289-300.

[6] Borisov N., Brumley D. et al. Generic Application-Level Protocol Analyzer and its Language. In Proc. of the Network and Distributed System Security Symposium, 2007, 16 p.

[7] Hopcroft J.E., Motwani R., Ullman J.D. Introduction to Automata Theory, Languages, and Computation. 3-е изд. Pearson, 2006, 560 p.

[8] Knuth D.E. Semantics of context-free languages. Mathematical systems theory, vol. 2, issue 2, 1968, pp. 127-145.

[9] Ford B. Parsing expression grammars: a recognition-based syntactic foundation. ACM SIGPLAN Notices, vol. 39, issue 1, 2004, pp. 111-122.

[10] Jim T., Mandelbaum Y., Walker D. Semantics and Algorithms for Data-Dependent Grammars. In Proc. of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2010, pp. 417–430.

[11] Afroozeh A., Izmaylova A. Iguana: A Practical Data-Dependent Parsing Framework. In Proc. of the 25th International Conference on Compiler Construction, 2016, pp. 267-268.

[12] Earley J. An Efficient Context-Free Parsing Algorithm. Communications of the ACM, vol. 13, issue 2, 1970, pp. 94-102.

[13] Jim T., Mandelbaum Y. A New Method for Dependent Parsing. In Proc. of the 20th European Conference on Programming Languages and Systems, 2011, pp. 378-397.

[14] Ganty P., Köpf B., Valero P. A Language-Theoretic View on Network Protocols. Lecture Notes in Computer Science, vol. 10482, 2017, pp. 363-379.

[15] Peach: a fuzzing framework which uses a DSL for building fuzzers and an observer based architecture to execute and monitor them. URL: https://github.com/MozillaSecurity/peach.

[16] Netzob: Protocol Reverse Engineering, Modeling and Fuzzing. URL: https://github.com/netzob/netzob.

[17] Sommer R., Amann J., Hall S. Spicy: a unified deep packet inspection framework for safely dissecting all your data. In Proc. of the 32nd Annual Conference on Computer Security Applications, 2016, pp. 558-569.

[18] Fisher K., Mandelbaum Y., Walker D. The next 700 data description languages. ACM SIGPLAN Notices, vol. 4, issue 1, 2006, pp 2-15.

[19] Fisher K., Gruber R. PADS: a domain-specific language for processing ad hoc data. ACM SIGPLAN Notices,vol. 40, issue 6, 2005, pp. 295-304.

[20] boofuzz: Network Protocol Fuzzing for Humans. URL: https://github.com/jtpereyda/boofuzz/.

[21] GitLab Protocol Fuzzer Community Edition. URL: https://gitlab.com/gitlab-org/security-products/protocol-fuzzer-ce.

[22] 010 Editor - Pro Text/Hex Editor. URL: https://www.sweetscape.com/010editor/.

[23] GNU poke, an extensible editor for structured binary data. URL: 10.5446/46118.

[24] Соловьев М.А., Бакулин М.Г. и др. Практическая абстрактная интерпретация бинарного кода. Труды ИСП РАН, том 32, вып. 6, 2020 г., стр. 101-110 / Solovev M.A., Bakulin M.G. et al. Practical abstract interpretation of binary code. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 6, 2020, pp. 101-110 (in Russian). DOI: 10.15514/ISPRAS-2020-32(6)-8.

[25] Соловьев М.А., Бакулин М.Г. и др. О новом поколении промежуточных представлений, применяемых для анализа бинарного кода. Труды ИСП РАН, том 30, вып. 6, 2018 г., стр. 39-68 / Solovev M.A., Bakulin M.G. et al. Next generation intermediate representations for binary code analysis. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 6, 2018, pp. 39-68 (in Russian). DOI: 10.15514/ISPRAS-2018-30(6)-3.

[26] Cousot P., Cousot R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Proc. of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 1977, pp. 238-252.

Приложение 1. Алгоритм разбора данных в соответствии с предложенной моделью (псевдокод)

```
     ВХОД: указатель на данные (экземпляр формата и адрес начала)
     ВЫХОД: структура результата Data

1. Сформировать множество неразобранных связей
2. Взять новую связь из множества неразобранных
3. Тип связи:
 - внутренняя или внешняя:
     3.1. Определить вычислимость положения
      - Не вычислимо: перейти к п.2
      - Вычислимо: вычислить -> POS
     3.2. Если POS = None, пометить связь как разобранную и
          перейти к п.2
     3.3. Определить вычислимость экземпляра формата:
      - Не вычислим: перейти к п.2
      - Вычислим: вычислить -> FORMAT
     3.4. Тип связи:
      - внутренняя:
          3.4.1. Для (POS, FORMAT) вызвать Алгоритм
          3.4.2. Полученный результат разбора добавить в структуру Data
          3.4.3. Пометить связь как разобранную
      - внешняя:
          3.4.4. Добавить (POS, FORMAT) в структуру Data
          3.4.5. Пометить связь как разобранную
      - связь-значение:
     3.5. Определить вычислимость значения:
      - Не вычислимо: перейти к п.2
      - Вычислимо:
          3.5.1. Вычислить -> VALUE
          3.5.2. Добавить VALUE в структуру Data
          3.5.3. Пометить связь как разобранную
4. Если еще есть связи в множестве неразобранных, перейти к п.2
5. Если ни одна связь не была разобрана, вернуть ОШИБКУ
6. Перейти к п.1
```

Информация об авторах / Information about authors

Александр Александрович ЕВГИН – аспирант ИСП РАН. Его научные интересы включают теорию формальных языков, форматы данных, сетевые протоколы, анализ бинарного кода.

Alexander Aleksandrovich EVGIN is a postgraduate at the system programming department. His research interests include formal language theory, data formats, network protocols, binary code analysis.

Михаил Александрович СОЛОВЬЕВ – кандидат физико-математических наук, старший научный сотрудник отдела компиляторных технологий ИСП РАН; старший преподаватель

кафедры системного программирования факультета ВМК МГУ. Его научные интересы включают анализ бинарного и исходного кода, обратную инженерию ПО, операционные системы.

Mikhail Aleksandrovich SOLOVEV is a candidate of physical and mathematical sciences, senior researcher at the compiler technologies department of ISP RAS; senior lecturer at the system programming department of the faculty of Computational Mathematics and Cybernetics of MSU. His research interests include binary and source code analysis, software reverse engineering, and operating systems.

Вартан Андроникович ПАДАРЯН – кандидат физико-математических наук, ведущий научный сотрудник отдела компиляторных технологий ИСП РАН; доцент кафедры системного программирования факультета ВМК МГУ. Его научные интересы включают компиляторные технологии, безопасность ПО, анализ бинарного кода, параллельное программирование, эмуляцию и виртуализацию.

Vartan Andronikovich PADARYAN is a candidate of physical and mathematical sciences, leading researcher at the compiler technologies department of ISP RAS; associate professor of the system programming department of the faculty of Computational Mathematics and Cybernetics of MSU. His research interests include compiler technologies, software security, binary code analysis, parallel programming, emulation, and virtualization.

DOI: 10.15514/ISPRAS-2021-33(6)-4

Использование идентификации потоков выполнения при решении задач полносистемного анализа бинарного кода

[1] *И.А. Васильев, ORCID: 0000-0003-3824-2753 <vasiliev@ispras.ru>*
[2] *П.М. Довгалюк, ORCID: 0000-0003-2483-5718 <pavel.dovgaluk@ispras.ru>*
[1] *М.А. Климушенкова, ORCID: 0000-0001-6737-9092 <maria.klimushenkova@ispras.ru>*
[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Новгородский государственный университет имени Ярослава Мудрого, 173003, Россия, г. Великий Новгород, ул. Большая Санкт-Петербургская, д. 41*

Аннотация. При полносистемном анализе бинарного кода зачастую применяется динамический бинарный анализ, при котором предоставляемый аналитику объем данных представлен потоком выполняемых инструкций и содержимым оперативной памяти и регистров. Для обработки таких данных требуется глубокое понимание особенностей исследуемой системы, при этом трудозатраты на выполнение анализа и требования к технической осведомленности пользователя становятся очень велики. Для упрощения процесса анализа необходимо привести входные данные к более дружелюбному для пользователя виду, т.е. предоставить высокоуровневую информацию об исследуемом программном обеспечении. Такой высокоуровневой информацией является информация о потоке выполнения программы. Для восстановления потока выполнения программы важно иметь представление о вызываемых ею процедурах. Получить такое представление можно с помощью стека вызова функций для конкретного потока. Построение стека вызовов без информации о выполняемых потоках невозможно, т.к. каждому потоку однозначно соответствует один стек и vice versa. Помимо этого, само наличие информации о потоках повышает уровень знаний о системе, позволяет более тонко профилировать объект исследования и проводить узконаправленный анализ, применяя принципы выборочного инструментирования. Виртуальная машина не предоставляет напрямую такой информации, и строить предположения о работе исследуемой системы приходится, основываясь на доступных низкоуровневых данных (поток выполняемых виртуальным процессором инструкций и оперативная память виртуальной машины). Таким образом, существует необходимость в разработке метода для автоматической идентификации потоков в исследуемой системе, опирающегося на имеющемся объеме данных. В данной работе рассматриваются существующие подходы к реализации получения высокоуровневой информации при полносистемном анализе и предлагается метод для восстановления данных о потоках в условиях полносистемной эмуляции с низкой степенью ОС-зависимости. Также приводятся примеры практического использования данного метода при реализации инструментов анализа, а именно: восстановление стека вызовов, обнаружение подозрительных операций возврата и обнаружение обращений к освобожденной памяти в стеке. Приведенное в статье тестирование показывает, что накладываемое описанными алгоритмами замедление позволяет проводить работу с исследуемой системой, а сравнение с эталонными данными подтверждает корректность получаемых алгоритмами результатов.

Ключевые слова: полносистемный анализ; стек вызовов; обнаружение уязвимостей

Для цитирования: Васильев И.А., Довгалюк П.М., Климушенкова М.А. Использование идентификации потоков выполнения при решении задач полносистемного анализа бинарного кода. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 51-66. DOI: 10.15514/ISPRAS–2021–33(6)–4

Using the identification of threads of execution when solving problems of full-system analysis of binary code

[1] *I.A. Vasiliev, ORCID: 0000-0003-3824-2753 <vasiliev@ispras.ru>*
[2] *P.M. Dovgalyuk, ORCID: 0000-0003-2483-5718 <pavel.dovgaluk@ispras.ru>*
[1] *M.A. Klimushenkova, ORCID: 0000-0001-6737-9092 <maria.klimushenkova@ispras.ru>*
[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Yaroslav-the-Wise Novgorod State University, 41, B. Sankt-Peterburgskaya st., Novgorod, 173003, Russia*

Abstract. Dynamic binary analysis, that is often used for full-system analysis, provides the analyst with a sequence of executed instructions and the content of RAM and system registers. This data is hard to process, as it is low-level and demands a deep understanding of studied system and a high-skileed professional to perform the analysis. To simplify the analysis process, it is necessary to bring the input data to a more user-friendly form, i.e. provide high-level information about the system. Such high-level information would be the program execution flow. To recover the flow of execution of a program, it is important to have an understanding of the procedures being called in it. You can get such a representation using the function call stack for a specific thread. Building a call stack without information about the running threads is impossible, since each thread is uniquely associated with one stack, and vice versa. In addition, the very presence of information about flows increases the level of knowledge about the system, allows you to more subtly profile the object of research and conduct a highly focused analysis, applying the principles of selective instrumentation. The virtual machine only provides low-level data, thus, there is a need to develop a method for automatic identification of threads in the system under study, based on the available data. In this paper, the existing approaches to the implementation of obtaining high-level information in full-system analysis are considered and a method is proposed for recovering thread info during full-system emulation with a low degree of OS-dependency. Examples of practical use of this method in the implementation of analysis tools are also given, namely: restoring the call stack, detecting suspicious return operations, and detecting calls to freed memory in the stack. The testing presented in the article shows that the slowdown imposed by the described algorithms allows working with the system under study, and comparison with the reference data confirms the correctness of the results obtained by the algorithms.

Keywords: full-system instrumentation, call stack, vulnerabilities detection.

For citation: Vasiliev I.A., Dovgalyuk P.M., Klimushenkova M.A. Using the identification of threads of execution when solving problems of full-system analysis of binary code. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 51-66 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–4

1. Введение

Можно выделить принципиально разные подходы при получении данных о потоках, которые зависят от уровня проведения динамического бинарного анализа: уровень приложений и полносистемный уровень. Различные системы анализа уровня приложений (Pin [1], Valgrind [2]) легко получают доступ к информации об исследуемом процессе и потоках, так как выполняются в одной операционной системе с исследуемым приложением и могут, используя системные API, получить интересующую информацию от ОС. Но в этом случае инструментарий анализа делит одно адресное пространство с исследуемым приложением, а значит может быть обнаружен и скомпрометирован. По этой причине данный подход нельзя использовать при решении вопросов обеспечения безопасности.

При полносистемном анализе прямого доступа к высокоуровневой информации получить нельзя, поэтому различные инструменты предлагают решение этой проблемы. Например, Virtuoso [3] использует для реализации программу-агент, которая собирает интересующие данные, и впоследствии создает алгоритм на основании полученной трассы выполнения этой программы. DECAF [4,5] при анализе основывается на полях данных из структур ядра исследуемой операционной системы. PEMU [6] не использует ни программы-агенты, ни

структуры ядра, однако использует примитивные алгоритмы для определения высокоуровневой информации, дающие приблизительный результат и строго-ориентированные под архитектуру x86.

Описанным подходам свойственны недостатки, ограничивающие возможность их применения: невозможность использования в системах с закрытым исходным кодом, невозможность использования во встроенных системах, не поддерживающих загрузку новых программ, низкое качество получаемых данных, строгая направленность на определенные ОС.

Таким образом, для обеспечения универсальности и достаточной степени доверия к результатам работы конечного алгоритма, необходимо в первую очередь использовать методы анализа, функционирующие вне исследуемой системы, т.е. на уровне виртуальной машины, а также опираться на данные, являющиеся ОС-независимыми или находящиеся в прямом доступе.

Отсюда возникают требования, предъявляемые к методу идентификации потоков в ОС: во-первых, необходимо избежать загрузки программ-агентов в гостевую ОС, а значит весь функционал, обеспечивающий идентификацию потоков, должен располагаться на уровне виртуальной машины. Во-вторых, необходимо обеспечить универсальность метода, т.е. опираться либо на ОС-независимые данные, либо на данные из открытых официальных источников. В-третьих, метод должен быть реализуем для различных процессорных архитектур.

При реализации метода восстановления стека вызовов функций необходимо придерживаться описанных выше принципов. При этом также важно не опираться в реализации на соглашение о вызовах, так как оно не всегда содержит необходимую информацию об указателе фрейма, а значит не может универсально использоваться при восстановлении стека вызовов.

Соблюдение этих требований позволит получить метод, применимый как для открытых, так и для закрытых систем. При этом такой метод можно будет с минимальными изменениями портировать на ОС одного семейства и адаптировать для работы на новых архитектурах.

В последующих главах будет описан разработанный метод идентификации потоков в операционной системе на уровне виртуальной машины, без встраивания программ-агентов в исследуемую систему и без использования данных из структур ядра ОС. Идентификация потоков подразумевает установку надежного разделения между различными потоками с возможностью последующего использования этой информации для реализации алгоритмов анализа. Полученный метод переносим и адаптируем для новых конфигураций исследуемых систем.

Также в данной статье уделяется внимание практическому применению разработанного алгоритма для решения задач анализа, например, будет предложен метод для восстановления стека вызовов, не опирающийся на данные соглашения о вызовах. Также будут приведены примеры использования полученных данных для вопросов решения безопасности использования ПО (алгоритм обнаружения подозрительных возвратов из процедур и алгоритм обнаружения использования освобожденной памяти в стеке).

2. Существующие реализации восстановления данных

Среди существующих подходов нам будут интересны те, в которых решается задача идентификации потоков в ОС, и данная информация используется для дальнейшей реализации алгоритмов анализа.

Рассмотрим следующие показатели предлагаемых подходов:

- ОС-зависимости алгоритма;
- использование программ-агентов, загружаемых в систему;

- возможность обнаружения со стороны исследуемой системы;
- возможность использования в отсутствии исходного кода исследуемой системы.

Системы, реализующие анализ уровня процессов (такие как: Pin [1], Valgrind [2], DynamoRIO [7]), как правило представляют из себя виртуальные машины, внутри которых запускается рассматриваемое приложение. Им свойственен прямой доступ к любой интересующей информации о рассматриваемом процессе – легкость доступа к данным и их полнота обеспечивается за счет выполнения инструментария анализа в одной операционной системе с исследуемым приложением, а значит и наличии доступа к системному API. Однако в этом случае невозможно анализировать межпроцессное взаимодействие, анализировать ядро ОС. При этом инструменты характеризуются строгой ОС зависимостью и не обеспечивают необходимой в вопросах безопасности изоляции инструмента и предмета анализа.

Среди систем, реализующих полносистемный анализ, рассмотрим Virtuoso [3], TEMU [8], DECAF [4,5], PANDA [10], PyREBox [11], PEMU [6], IceBox [12].

Создатели инструмента Virtuoso [3] реализуют получение информации о системе извне, на уровне виртуальной машины, в которой запущена гостевая система. Однако для определения способа извлечения интересующей информации в данном инструменте применяется внедряемая программа-агент. Программа-агент определяет адреса интересующих данных в системе, что позволяет в последующем использовать их при выполнении анализа. Данный подход не может применяться при работе с самомодифицирующимся кодом, при работе с системами, реализующими ASLR, а само использования программ-агентов ограничивает область применения подхода до открытых систем.

TEMU [8] реализует динамическое бинарное полносистемное инструментирование на базе эмулятора QEMU [9]. Определенные события в гостевой системе вызывают функции обработки в подключаемых модулях, которые и выполняют анализ данных. Для получения информации о семантике уровня операционной системы используется “семантический извлекатель” - модуль, ответственный за извлечение информации о процессах, модулях, потоках, символьной информации и контексте выполнения. Данный модуль поддерживает операционные системы Windows XP, Windows 2000 и Linux, причем реализация получения данных из ОС для них отличается. Для Linux семантический извлекатель обращается напрямую к структурам данных ядра по предзаданным адресам. Для Windows используется специальный модуль ядра – он загружается в гостевую систему и выполняет функцию отслеживания специфических событий (запуск процессов, потоков, загрузка модулей) и передачи этих данных в саму инструментальную систему. Таким образом TEMU использует как строго ОС-специфичные данные, так и загрузку модуля в исследуемую систему, что является значительным минусом для инструмента.

DECAF [4,5] также является фреймворком динамического бинарного полносистемного инструментирования, причем его создатели при разработке опирались на опыт TEMU. Им удалось избавиться от необходимости использования загружаемого модуля, однако вся информация о процессах, потоках, модулях основывается на содержимом структур ядра исследуемой системы, а значит подход является строго ОС зависимым и для каждой итерации ОС и даже версии ядра будет требовать применения приемов обратной инженерии.

Схожий подход используют и фреймворки PANDA [10] и PyREBox [11]. Процесс анализа в обоих случаях базируется на конфигурационном файле, строго специфичном для ОС, который в случае с PANDA генерируется загружаемой в систему программой-агентом, а для PyREBox используются готовые конфигурационные файлы, заимствованные из Volatility [13]. При этом в результате для получения информации используется считывание данных из структур ядра исследуемой системы.

PANDA при этом предлагает также и способы условно ОС-независимые, но они дают результаты низкой точности: так, например, определение потоков по ASID дает ложные результаты, если одному процессу соответствует несколько потоков, а эвристический метод

основан на определении смены потока по резкой смене указателя стека, что не всегда соответствует действительности.

PEMU [6] является полносистемным продолжателем идей Pin – предоставляет тот же API, но расширяет доступную для исследования область применения. При этом разработчики предоставляют возможность получения данных о процессах и потоках из системы, т.к. без этой информации значительная часть API функций, переносимых из Pin не смогла бы функционировать. Однако решение строго привязано к специфике процессора x86 и для потоков является только ориентировочным: поток идентифицируется по значению регистра esp с наложенной обнуляющей маской на нижние 12 бит.

IceBox [12] – инструмент для интроспекции виртуальной машины в VirtualBox, позволяющий трассировать и отлаживать пользовательские процессы и процессы ядра в исследуемой системе. Основывается на Winbagility [14] от которой и наследует недостатки – строгая зависимость от инструментария WinDBG, возможность работы только с ОС семейства Windows и использование для анализа данных из структур ядра исследуемой системы.

Таким образом, были определены основные недостатки рассмотренных в обзоре инструментов.

- Использование данных структур ядра. Для систем с закрытым исходным кодом получение данной информации сопряжено или со сложным анализом системы, или вовсе невозможно. Также данное решение является строго ОС-специфичным, а значит требует генерации новой конфигурации для каждой из исследуемых систем, вплоть до версий и сервис паков.
- Использование программ-агентов. Многие из рассмотренных инструментов используют программы-агенты для получения данных из системы напрямую, либо для генерации файлов-конфигураций, используемых при последующем анализе. Использование таких внедряемых программ невозможно для неизменяемых образов исследуемых систем. В случае, если внедрение возможно, исследуемая система может изменить свое поведение. Кроме того, генерация файлов конфигураций будет бесполезна в случае, если расположение структур данных определяется случайным образом при каждой загрузке системы.
- Сложность портирования и сопровождения в связи с ориентированностью решения на определенную операционную систему.

3. Предлагаемый метод восстановления данных

Предлагаемый нами подход для восстановления информации о потоках лишен недостатков, обнаруженных при рассмотрении других инструментов, и позволяет получать информацию при полносистемном динамическом анализе, не используя программы-агенты. Принципы, на которых построен разработанный метод, также обеспечивают простоту дальнейшей поддержки и адаптации для новых систем.

Суть предлагаемого метода идентификации потоков заключается в следующем. Известно, что в системе выполняются процессы, и каждому процессу соответствует один или более потоков [15]. Для организации виртуальной памяти система выделяет для каждого процесса новое значение записи в таблице трансляции виртуальных адресов в физические. Процессор выделяет отдельный регистр для хранения этого значения (для x86 это CR3, для ARM - CP15 и т.п.). Таким образом, наблюдая за значением этого регистра, можно однозначно отличать друг от друга запущенные процессы. В рамках процесса может выполняться несколько потоков, и каждый из потоков использует свои локальные переменные и оперирует своими адресами возврата.

Для организации работы потоков система выделяет каждому потоку отдельный стек. Значение верхушки стека (SP, stack pointer) хранится в определенном регистре процессора

(x86 - ESP, ARM - R13). Таким образом, идентифицировать поток можно по процессу, к которому он принадлежит, и по выделенному ему стеку. Однако, определение выделенного стека является нетривиальной задачей, в отличие от определения процесса. Для идентификации процесса достаточно отслеживать состояние регистра, содержащего значение записи в таблице трансляции. Изменения же в регистре, содержащем значение вершины стека, не всегда говорят о смене потока, так как стек постоянно изменяется и при нормальной работе потока: на него помещаются значения переменных, значения адресов возврата, он постоянно расширяется и сужается. При этом очевидно, что раз у каждого потока стек должен быть свой, то и значение регистра с вершиной стека для каждого потока будет изменяться только в рамках определенного диапазона и не будет пересекаться между разными потоками.

Отсюда следует вывод, что для однозначной идентификации потока необходимо и достаточно рассматривать пару “идентификатор процесса” и “диапазон значений стека”. На рис. 1 визуально отображено соотношение идентификатора потока и системной информации.

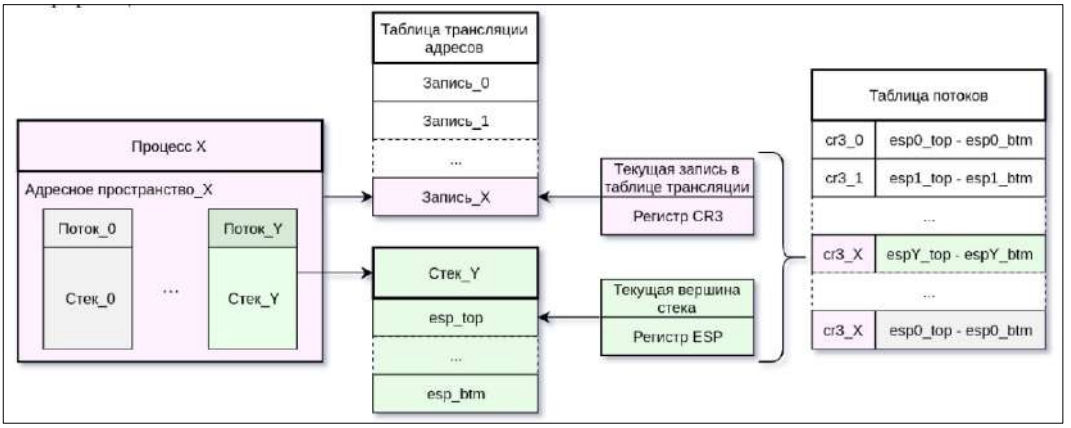


Рис. 1. Визуализация предлагаемого метода идентификации потоков
Fig. 1. Visualisation of proposed method of threads identification

Поскольку задача определения процесса является тривиальной, сложность метода сводится к задаче определения диапазона значений стека – как понять, относится ли очередное значение SP к тому же потоку или созданному новому? Предлагаемый метод основывается на предположении, что в системе должен присутствовать определенный набор событий, предшествующих операции создания нового стека и связанного с ним потока. Другие же события, связанные с изменением стека, служат для расширения границ диапазонов уже обнаруженных стеков.

Реализация разработанного метода строится на базе эмулятора с открытым исходным кодом QEMU [9]. Для внесения дополнительного функционала эмулятор поддерживает систему плагинов – динамических загружаемых библиотек, написанных на языке C. При этом источником данных для обработки плагином может являться как эмулятор, так и другие плагины. Это позволяет организовать комплексные алгоритмы анализа, где одни плагины опираются на данные, восстановленные другими плагинами.

Типичный процессор имеет ряд инструкций явно и косвенно взаимодействующих со стеком, а значит изменяющих указатель на его вершину. Например, к явно изменяющим относятся инструкции, использующие SP как операнд, к косвенно – выполняющие функционал push и pop (помещение значения на стек, и получение значения с вершины стека), инструкции вызова и возврата из функций.

Анализируя выполняющиеся при работе исследуемой системы инструкции и изменение значения SP, можно выделить подмножество инструкций и событий, появление которых свидетельствует о создании нового потока, и подмножество инструкций, расширяющих

существующий диапазон. Данные подмножества являются архитектурно-зависимыми, однако, как было упомянуто выше, имеют схожий вид для различных процессорных архитектур. При этом такой способ определения потоков является ОС-независимым. Применение данного метода обеспечивает легкость переноса на новые архитектуры и новые операционные системы.

Нами был разработан плагин, реализующий разбиение значений стека на диапазоны и идентификацию потоков. Логика работы разработанного плагина отображена на рис. 2 и заключается в следующем: при трансляции инструкции QEMU передает сигнал в плагин, где по опкоду определяется её тип и принадлежность к группе выделенных событий. При возникновении события, потенциально расширяющего текущий диапазон, происходит сравнение нового значения SP с границами текущего диапазона, и при необходимости его границы расширяются соответствующим значением. При возникновении события, потенциально создающего новый диапазон, происходит проверка вхождения нового значения вершины стека в один из существующих диапазонов. Если ни к одному из существующих диапазонов новое значение не принадлежит, то это свидетельствует о создании нового диапазона с полученным значением SP. Так как возможно ложное создание нового диапазона, то также предусмотрен механизм объединения, если в процессе расширения диапазона произошло наложение на соседний. В таком случае, один из диапазонов удаляется, а второй расширяется до объединенных границ. При этом происходит генерация сигнала об объединении, сопровождаемого идентификаторами потоков, чтобы позволить опирающимся на данные о потоках алгоритмам анализа провести соответствующее обновление данных.

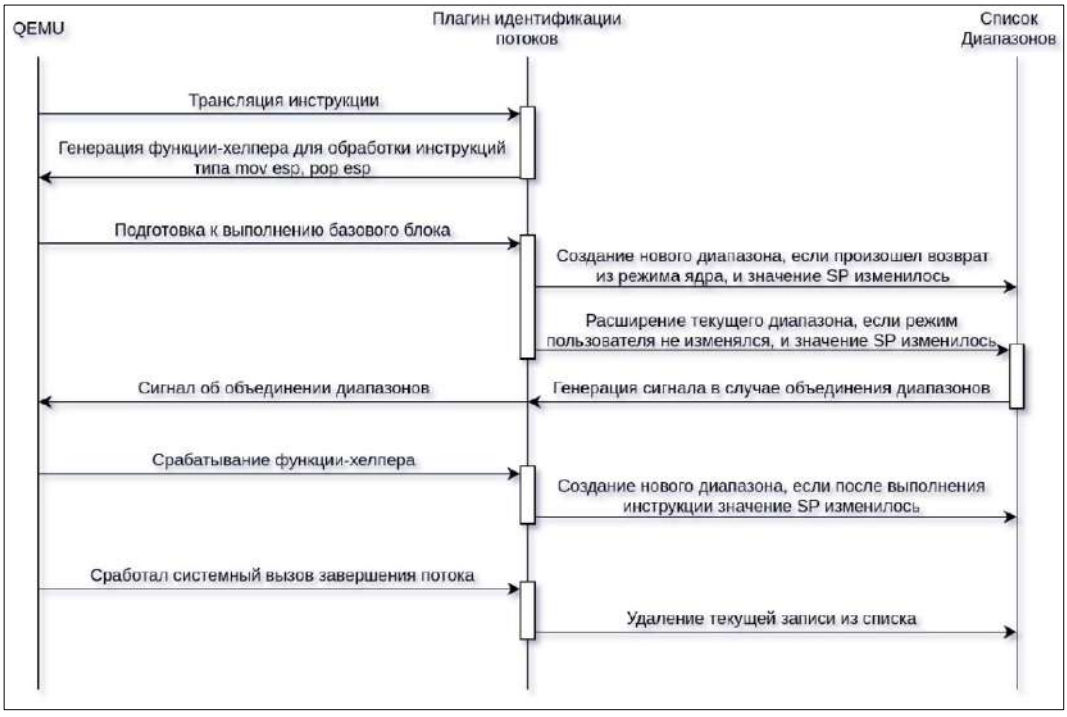


Рис. 2. Логика работы плагина идентификации потоков
Fig. 2. How the thread identification plugin works

Для плагина идентификации потоков были определены следующие условия для создания и расширения диапазонов (подробнее процесс определения условий описан в [16]). Для потенциального создания нового диапазона в i386 и x86_64: возникновение инструкции pop

esp, mov esp и возвращение из режима ядра в пользовательский режим, при этом новый диапазон будет создан, только если значение SP не входит в один из существующих. Отдельно стоит заметить, что при работе с ОС семейства Windows приходится также учитывать существование системного вызова NtContinue, который изменяет значение SP в режиме ядра, однако данное событие соответствует или расширению текущего потока, или переключению в другой существующий.

Для создания нового диапазона в mips64: возникновение инструкций move sp, lw sp, lui sp, ld sp, add sp, или инструкций addu sp, addiu sp, sub sp, subu sp, где один из аргументов при выполнении арифметической операции является нулем, или инструкций типа uld и uwd.

Описанный в [16] алгоритм был технически доработан для ускорения работы. В первую очередь использование тяжеловесного capstone было заменено на самописный дизассемблер под конкретные задачи для интересующих архитектур. Также хранение диапазонов значений указателя стека было оптимизировано. Ранее диапазоны объединялись в случае пересечения только при запросе данных у плагина. Так как для работы других плагинов анализа эта информация запрашивается с высокой периодичностью, то возникла потребность объединять диапазоны сразу в момент расширения.

Для упрощения этого процесса хранение диапазонов было приведено к упорядоченному списку, где в случае расширения и наложения диапазонов достаточно выполнить объединение с соседними, и не проверять остальные диапазоны. Дополнительно был изменен и момент обработки событий – ранее обработка выполнялась сразу после возникновения выделенных инструкций, теперь же значение SP проверяется только перед выполнением очередного блока, но с учетом произошедших в прошлом блоке событий. Суммарно введенные усовершенствования изменили время, проводимое в обработчике событий в плагине, с 20% до 0,5% от всего времени выполнения эмулятора (замер производился с помощью perf).

Отдельно стоит отметить, что для потоков существует необходимость в отдельной обработке их завершения (что также отображено на рис. 2). В общем случае, каждый поток однозначно идентифицируются описанной выше парой значений: “идентификатор процесса” и “диапазон значений стека”. Этих данных достаточно, чтобы отличать потоки между собой и реализовывать на основании этой информации дальнейшие алгоритмы анализа и инструментирования. Однако после завершения работы потока существует вероятность, что при создании нового потока система использует ту же область памяти, что и для завершенного потока. Для некоторых алгоритмов различие этих потоков будут критичны (тонкие алгоритмы профилирования, отладка потоков). В таком случае требуется ввести обработку системных вызовов.

Известно, что при завершении потока ОС генерирует соответствующий системный вызов, и определить это событие можно отслеживая поток выполняемых инструкций и значения регистров. Обнаружив соответствующий системный вызов, текущий диапазон необходимо отметить “завершенным” и оповестить об этом все зависимые алгоритмы анализа. Информация о системных вызовах является открытой и широко доступной для большинства операционных систем. Таким образом, если возникает необходимость в использовании системных вызовов, это добавляет методу фактор ОС-зависимости, однако позволяет сохранить простоту портирования за счёт открытости рассматриваемых данных.

4. Практическое применение восстановленных данных

4.1 Стек вызовов

Имея данные о потоках, восстановленные описанным выше плагином, можно приступить к реализации механизмов для анализа исследуемой системы. Одним из широко применяемых при анализе и отладке инструментов является стек вызовов. В данном случае речь ведется о

восстановлении стека вызовов в условиях полносистемного бинарного анализа, а значит алгоритм должен быть ОС-независимым и также работать как для функций пользовательского уровня, так и для уровня ядра. При реализации алгоритма важно не опираться на соглашение о вызовах, т.к. информация об указателе фрейма не всегда описана в соглашении, а значит и восстановить стек вызова с его помощью в этом случае не удастся.

Логика работы алгоритма восстановления стека вызовов заключается в следующем: из выполняемых в гостевой машине инструкций выделяются те, что относятся к вызову функции, и те, что свидетельствуют об операции возврата. Для каждого из потоков сохраняется свой список адресов, по которым произошел вызов, а при возникновении возврата соответствующая запись из списка вызовов удаляется.

Изначально в качестве анализируемой данным плагином единицы выступал блок трансляции. На этапе трансляции в блоке проверялась последняя инструкция, и блок помечался, если это была инструкция типа “вызов” или “возврат”. На этапе выполнения после блоков “вызов” в стек вызовов заносился адрес последней инструкции и ожидаемый адрес возврата. Перед выполнением блока, идущего за блоком “возврат” происходила проверка соответствия адреса первой инструкции в блоке верхнему адресу возврата в текущем стеке вызовов, и происходило удаление записи, в случае совпадения.

Однако из-за присутствующей в QEMU оптимизации, при которой идущие подряд блоки объединяются в цепь для последовательного связанного выполнения, событие “после выполнения блока” вызывается лишь для последнего блока в цепи. По этой причине логика плагина была изменена таким образом, что ключевыми событиями при анализе стали “трансляция инструкции” для обнаружения интересующих вызовов и возвратов, и “перед выполнением блока” для уточнения дополнительных данных о вызове и удаления записей после успешного возврата.

В результате алгоритм принял следующий вид: на этапе трансляции плагин получает сигнал от QEMU с информацией о транслируемой инструкции и определяет её тип по опкоду с помощью специального разборщика. Выделяются инструкции типа call и ret. Если была обнаружена одна из таких инструкций, то для нее создается специфическая функция-хелпер – call_helper и ret_helper. Данные функции-хелперы вызываются на этапе выполнения кода перед соответствующей инструкцией. Вызов перед выполнением, а не после является вынужденной мерой, так как особенности эмулятора QEMU не позволяют размещать хелперы после крайней инструкции в блоке трансляции, каковыми чаще всего и являются call и ret.

В call_helper происходит запрос текущего идентификатора из плагина идентификации потоков, и по данному идентификатору в таблице стеков вызовов обновляется соответствующий стек – добавляется новая запись с адресом вызова функции равным адресу инструкции call и соответствующим адресом на системном стеке, где будет сохранено значение адреса возврата после выполнения этой функции (хоть сам адрес возврата на этом этапе еще не был записан в стек, его расположение в стеке можно однозначно предсказать уже на этом этапе). Также происходит переключение глобальной переменной was_call, информация о чем будет использоваться на этапе обработки выполнения следующего блока. Для ret_helper происходит переключение глобальной переменной was_ret.

На этапе выполнения также происходит обработка сигнала от QEMU “перед выполнением блока трансляции”. Если была установлена отметка о произошедшем в конце предыдущего блока call, то на данном этапе к соответствующей записи в стеке вызовов происходит добавление данных об адресе вызванной процедуры (адрес первой инструкции текущего блока) и адрес возврата из текущей функции, извлекаемый из стека. Затем происходит происходит проверка на наличие устаревших записей и их последующее удаление. Такие записи возникают из-за того, что библиотеки операционной системы зачастую пользуются нестандартными способами оперирования со стеком и процессом возврата из функции.

Например, адрес возврата может быть перезаписан на предустановленное значение, или может произойти мнимый “возврат” без фактического использования инструкций ret. В этом случае на вершине стека могут оказаться записи, возврат по которым уже произошел, но не был пойман алгоритмом, которые необходимо удалить. Проверка на то, что запись является неактуальной, выполняется следующим образом: в начале блока считывается текущее значение указателя на вершину стека. Затем, данное значение сверяется с записанными на этапе вызова адресами в стеке, содержащими адреса возврата из функций, так как для их использования они все еще должны быть доступны. Если текущая вершина стека располагается выше (для растущих вниз стеков) или ниже (для растущих вверх стеков), чем адрес возврата для текущей записи, то эта запись удаляется с вершины стека вызовов, так как этот адрес возврата.

Дальнейшая обработка происходит только при установленном переключателе was_ret, так как выполняет функции обработки возврата и приведения стека вызовов в актуальное состояние. На этом этапе происходит сравнение текущего адреса базового блока с адресами возврата, записанными в стеке вызовов. При этом важно учитывать, что возврат может осуществляться через несколько записей в стеке вызовов, эмпирическим путём было получено значение в 10 верхних записей, которые достаточно проверять чтобы покрыть такой случай. Если адрес блока совпал с адресом возврата, то эта и все вышестоящие записи из стека вызовов удаляются.

Отдельная ситуация, которая нуждается в специфической обработке – инструкция типа jmp иногда может выступать в качестве call. Определить это можно по началу следующего блока трансляции. Если он принимает классическую форму пролога функции с записью фрейма стека в стек, то это свидетельствует о произошедшем вызове. Чтобы учесть данную ситуацию, на этапе трансляции помимо инструкций типа call и ret необходимо также устанавливать и переключатель was_jmp для jmp инструкций. На это выполнения блока при установленном переключателе was_jmp происходит проверка первых инструкций данного блока на ожидаемую последовательность, и в случае совпадения происходит добавление соответствующей записи в стек вызовов.

Также необходимо учитывать, что идентификатор стека устанавливается в соответствии с информацией, полученной от плагина идентификации потоков. Как было упомянуто в предыдущем разделе, работа идентификатора потоков подразумевает, что диапазоны после создания могут объединяться, а значит это может привести к созданию нескольких стеков вызовов вместо одного в рассматриваемом плагине. Для того, чтобы избежать этой ситуации, плагин отслеживает возникновение сигнала “объединение диапазонов”, сопровождаемого идентификаторами диапазонов, что позволяет объединить соответствующие стеки вызовов в соответствие с направлением роста стека в данной системе.

4.2 Обнаружение подозрительных операций возврата

Также, на основе выше описанных алгоритмов был разработан и реализован метод обнаружения подозрительных операций возврата. Одним из используемых злоумышленниками способов повлиять на выполнение программы является передача управления на вредоносный код. Рассмотрим пример, когда функция после окончания выполнения совершает возврат, но попадает в заданный злоумышленником участок кода. Происходить это может по причине того, что значение адреса возврата в стеке было перезаписано с использованием ошибок в программе. Ключевым механизмом для реализации уязвимостей такого рода является использование переполнения буфера для подмены реального адреса возврата на ссылающийся на зловредный код (рис. 3). Такие уязвимости эксплуатируются как ROP-цепочками (цепочка "гаджетов" в коде, традиционно из нескольких инструкций, где каждый гаджет завершается инструкцией ret), так и используются для загрузки готового шеллкода в память для непосредственного выполнения

этих инструкций по нужному адресу. В любом случае, первым шагом в эксплуатации такой уязвимости является подмена адреса возврата, и именно на обнаружение такой манипуляции и направлен реализованный механизм.

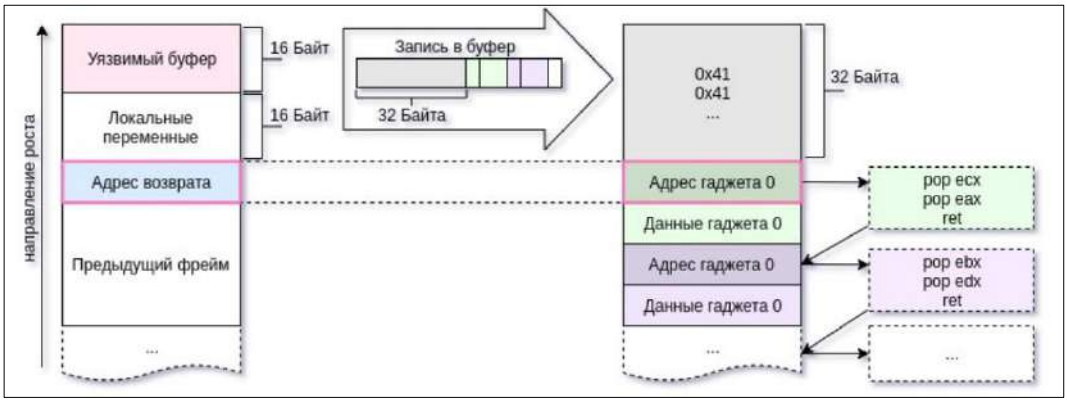


Рис. 3. Пример эксплуатации уязвимого буфера с помощью ROP-цепочки
Fig. 3. An example of exploitation of a vulnerable buffer with a help of a ROP-chain

При работе механизма пользователь получает оповещение каждый раз, когда после возврата из процедуры выполнение программы переходит не к ожидаемому адресу возврата, т.е. происходят так называемые “подозрительные операции возврата”. Функционал данного плагина строится на информации, получаемой из плагина восстановления стека вызовов. После каждой инструкции типа ret происходит генерация соответствующего сигнала, что позволяет описываемому плагину запросить информацию о текущем стеке вызовов и сравнить адрес вызванного базового блока с верхними записями в стеке вызовов.

Если совпадения обнаружено не будет, это является свидетельством неожиданного поведения программы и перехода в подозрительную часть кода. Это может может являться признаком зловредного характера происходящих действий в программе, поэтому в данной ситуации плагин выводит для пользователя информацию об адресе выполненной инструкции ret, ожидаемом адресе возврата и фактическом адресе возврата. Обладая этой информацией, пользователь может проверить вызванный код, чтобы определить характер и потенциальную опасность происходящей операции.

Так как функционал алгоритма построен на принципах полносистемного бинарного анализа, это позволяет выполнять обнаружение зловредного воздействия даже на уровне ядра, что недоступно широко используемым программам для автоматического анализа пользовательских приложений (таким как Valgrind). Однако во время анализа инструкций, относящихся к выполнению системных библиотек, было обнаружено, что ОС может намеренно использовать модификацию адреса возврата в стеке для прямого вмешательства в поток выполнения кода и перехода на некоторые утилитарные функции.

С точки зрения разработанного механизма данные переходы также будут выглядеть как “подозрительные” возвраты. По этой причине сообщения, предоставляемые плагином аналитику, должны использоваться не для совершения выводов о характере поведения программы, а в качестве указателя на участки кода, которые пользователь сможет проанализировать и в дальнейшем самостоятельно дать им характеристику.

Отдельно учитывается такая ситуация, как использование инструкции ret в качестве инструкции jmp – программа помещает в стек значение адреса для последующего перехода, затем вызывает ret и таким образом осуществляет безусловный переход, при этом фактически использование ret в данном случае не является операцией возврата. Для того, чтобы подобный механизм не вызывал срабатывание алгоритма плагина, была добавлена дополнительная проверка – при выполнении инструкции вызова запоминается адрес в стеке,

по которому записан адрес возврата. При последующем выполнении инструкции ret проверяется адрес в стеке, из которого было извлечено значение для возврата. Если этот адрес отличается от записанного при вызове, то это является свидетельством использования инструкции ret в качестве jmp.

Таким образом подобные инструкции не учитываются алгоритмом и удается избежать ложного оповещения аналитика. Данный механизм был включен в плагин восстановления стека вызовов, что позволяет поддерживать корректность восстановленных данных и генерировать сигнал о событии (инструкции возврата) только для фактических операций возврата.

В данном плагине поддерживается возможность указания идентификатора потока, для анализа только конкретной интересующей программы, а не всего процесса работы системы. Это может быть полезно для сепарации системных библиотек, содержащих корректные "подозрительные" возвраты, от интересующих аналитика программ. Такой подход позволяет минимизировать замедление работы исследуемой системы и упростить последующий процесс анализа за счет первичного отсеивания полезных данных.

4.3 Обнаружение обращений к освобожденной памяти

Также, базируясь на восстановленных данных о потоках, был реализован плагин, который отслеживает обращения к памяти в стеке за пределами текущих границ стека (частный случай use after free). Алгоритм функционирует следующим образом: QEMU генерирует сигналы о работе с памятью, и на каждый сигнал, соответствующий загрузке из памяти в данном плагине назначается функция-обработчик. В данной функции происходит запрос информации о текущем потоке от плагина восстановления данных о потоках. Из полученной информации интерес представляют границы возможных значений SP для выполняющегося потока. Также происходит запрос значения регистра, содержащего текущее значение SP, от QEMU.

Адрес чтения из памяти сравнивается с полученными значениями. Если данный адрес располагается в сторону роста стека, относительно значения вершины стека, но при этом попадает в диапазон адресов, принимаемых SP, для текущего потока, то можно сделать однозначный вывод о попытке загрузки информации из адресов, потерявших актуальность. Сообщение о такой загрузке будет выведено в лог и представлено пользователю. При такой загрузке есть большая вероятность получить непредсказуемое поведение программы, что может как быть источником трудно отлаживаемых ошибок, так и уязвимым местом для зловредного воздействия на программу.

Данный алгоритм не всегда применим для системных библиотек, так как при их работе было может встретиться преднамеренное использование данных из стека, лежащих выше текущей вершины стека. Этот подход нельзя назвать хорошей практикой, однако он не является редкостью и неоднократно встретился при анализе работы различных ОС. Авторы ПО в данном случае предполагают, что они имеют полный контроль над стеком, знают его содержимое в любой момент и обращение такого характера к данным не может вызвать непредвиденных ошибок. Однако, даже несмотря на подобные "ложные" срабатывания, результат работы плагина (т.е. лог обращений к адресам за пределами стека) может использоваться для анализа работы программ, как элемент отладки, а также для улучшения надежности разрабатываемого ПО.

В результате был реализован ряд связанных взаимодействующих плагинов (рис. 4), где каждый из последующих плагинов применяет данные, полученные в результате работы предыдущего.

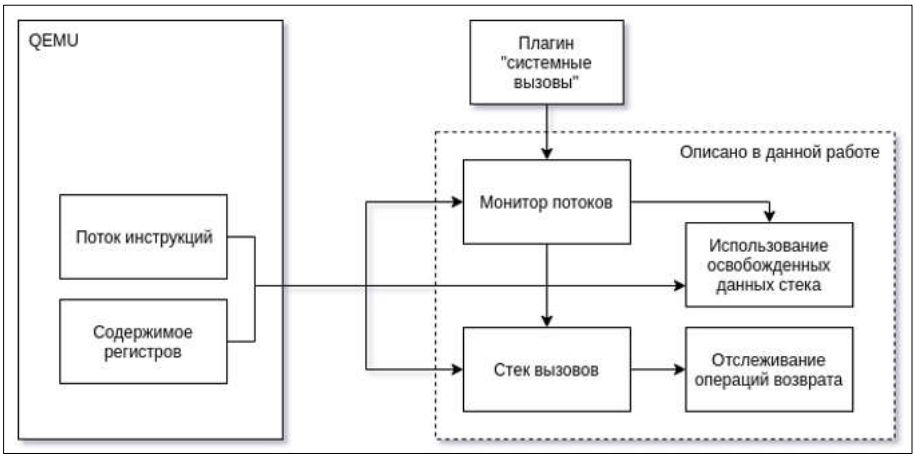


Рис. 4. Схема взаимосвязи реализованных плагинов и эмулятора
Fig. 4. The diagram of the relationship between the implemented plugins and the emulator

5. Оценка замедления и корректности

Для проведения сравнения производительности и функциональности в данной главе рассматриваются следующие комбинации архитектур и операционных систем: i386 и Windows XP, x86_64 и Windows 7, mips64 и операционная система реального времени (ОС РВ).

Для оценки накладываемого замедления были рассмотрены следующие сценарии:

- QEMU без подключения разработанных плагинов;
- предыдущий сценарий с дополнительно подключенным плагином для идентификации потоков thread_monitor;
- предыдущий сценарий с дополнительно подключенным плагином для восстановления стека вызовов call_stack.

При этом во время тестирования в системах выполнялись следующие операции:

- для i386 + Windows XP рассматривалась загрузка операционной системы, запуск и выполнение программы Solitaire.exe;
- для x86_64 + Windows 7 рассматривалась загрузка операционной системы, запуск и выполнение программы Paint.exe;
- для Mips64 + ОС РВ рассматривалась загрузка операционной системы, и многократное выполнение операций show processes и stacktrace.

Для каждого сценария проводилось не менее пяти замеров времени выполнения, после чего крайние значения были отброшены, а по оставшимся было вычислено среднее время выполнения сценария и полученное замедление.

Измерение времени производилось с помощью встроенной в Ubuntu утилиты time, при этом для достижения более точного результата замер времени производился не при непосредственной работе системы, а при воспроизведении записанного журнала выполнения. Таким образом каждый из запусков сценарий был полностью идентичен, что позволят говорить о корректном сравнении времени выполнения. Результаты исследования приведены в табл. 1. Наибольшее замедление во всех сценариях соответствует использованию плагина по восстановлению данных стека вызовов, что оправданно, так как данный случай помимо собственных вычислений включает и два предыдущих сценария. При этом самое большое среднее замедление было показано в исследуемой системе x86_64 + Windows 7 (55%). Данное замедление является ощутимым, однако оно не мешает работе с системой, комфортному

использованию большинства приложений и позволяет осуществлять практически любые необходимые при анализе операции. Также можно заметить, что данное замедление является относительно небольшим, в сравнении с аналогичными фреймворками анализа, базирующимися на инструментировании. Например, авторы PEMU в своей работе [6] приводят следующие данные по замедлению, накладываемому различными плагинами: минимальное в 149%, максимальное в 922%, со средним значением в 433%.

Табл. 1. Замедление, оказываемое разработанными плагинами на работу QEMU
Table 1. Slowdown caused by implemented plugins

Исследуемая система	QEMU, мин	thread_monitor, мин	call_stack, мин
i386 + WindowsXP	3:40	3:44	3:53
Замедление		1,8%	5,9%
x86_64 + Windows7	4:43	5:59	7:19
Замедление		27%	55%
mips64 + ОС РВ	1:21	1:24	1:33
Замедление		3,7%	15%

Для определения корректности получаемых результатов для разных систем и плагинов были использованы различные методы.

Для проверки корректности работы алгоритма идентификации потоков в операционных системах семейства Windows было выполнено сравнение распознанных потоков с содержимым системной структуры TEB, а именно с полем UniqueThread. Адреса необходимых структур были получены средствами обратной инженерии. Для 32-битной Windows XP адрес структуры TEB располагается по смещению 0x18 относительно регистра FS, а поле UniqueThread располагается в TEB по смещению 0x24. Для 64-битной Windows 7 адрес структуры TEB располагается по смещению 0x30 относительно регистра GS, а поле UniqueThread располагается в TEB по смещению 0x48.

При сравнении было выявлено, что каждому идентифицированному потоку в плагине thread_monitor соответствует один идентификатор из системной структуры, при этом не возникает ситуаций, когда один идентификатор из структуры соответствует нескольким идентифицированным потокам. В редких случаях, когда система завершает работу потока без использования соответствующего системного вызова, возникает ситуация с одним идентифицированным потоком и соответствующими несколькими системными идентификаторами. Зачастую это не играет роли для дальнейшего анализа (например, это не сказывается на корректности возвращаемых данных плагином восстановления стека вызовов), однако необходимо учитывать эту особенность при дальнейшей проектировке плагинов анализа.

Для определения корректности работы алгоритма восстановления стека вызовов в ОС семейства Windows были собраны тестовые программы, к которым удаленно подключался отладчик GDB. Далее выполнялось сравнение адресов, возвращаемых отладчиком по команде backtrace, с адресами, записанными в восстановленном плагином стеке. Полученные результаты показали, что плагин позволяет получить корректные данные.

Для системы mips64 с ОС РВ при определении корректности результата работы алгоритмов была возможность использовать системные команды: команда вывода списка потоков используется для сравнения с результатами идентификации потоков; команда вывода списка вызванных внутри потока функций используется для проверки восстановленного стека вызовов. В обоих случаях были получены корректные данные, восстановленный стек вызовов полностью соответствовал возвращаемому системой.

Для оценки работы плагина для обнаружения подозрительных операций возврата были рассмотрены два примера: синтетический, написанный для доказательства потенциальной

пользы использования алгоритма, и реальный, использующий существующее приложение с уязвимостью и описанный на агрегаторе уязвимостей (сайте exploit-db.com) зловредный код.

Синтетический пример представлял из себя тестовую программу, выполняющую ряд вычислений, но сопровождаемую ассемблерной вставкой, которая напрямую подменяет значение адреса возврата в стеке и выполняет инструкцию ret. Данный пример был успешно обнаружен плагином, что свидетельствует о потенциальной возможности его использования для защиты от зловредного кода.

Для реального примера обнаружения злонамеренной подмены адреса возврата была рассмотрена программа с уязвимостью gAlan, для 32-битных ОС Windows. В данной программе при загрузке пользовательских файлов не выполняется проверка размера, что может вызвать переполнение буфера, а значит идеально подходил для атаки злоумышленника. В качестве примера зловредного кода был использован код под номером 10345 из базы данных уязвимостей (exploit-db.com) – это скрипт на языке Python, который формирует пользовательский файл с необходимым содержимым, в результате загрузки которого в gAlan происходит переполнения буфера и выполняется запуск программы "калькулятор". Содержимым, записываемым скриптом в файл, помимо прочего, является мусор, чтобы заполнить стек вплоть до адреса возврата, новый адрес возврата, роль которого выполняет адрес инструкции call esi из библиотеки glib-1_3, распространяемой в комплекте с уязвимой программой, и шеллкод, который будучи помещенным в память и выполненным, вызовет запуск системной программы “калькулятор”.

При запуске данного зловредного кода плагин обнаружения подозрительных операций возврата благополучно оповестил аналитика о потенциальной атаке, при этом отобразив адрес инструкции возврата и адрес, на который произошел возврат. Данной информации достаточно, чтобы при дальнейшем анализе сделать выводы о зловредности по участку кода, на который осуществляется переход, а также чтобы определить, что изначально являлось источником уязвимости. Таким образом, плагин обнаружения подозрительных операций возврата показал свою действенность и в условиях реальной уязвимости и использования её злоумышленником.

6. Заключение

В данной работе был предложен метод восстановления данных о потоках и описана его реализация. Данный метод лишен ряда недостатков, присутствующих в аналогичных фреймворках, и может использоваться для широкого ряда операционных систем за счёт обеспечения низкой ОС-зависимости. Основная часть работы сконцентрирована на описании способов применения полученных данных о потоках для реализации более комплексных инструментов для аналитика: плагин восстановления стека вызовов, плагин, отслеживающий подозрительные операции возврата, плагин мониторинга обращения к освобожденной памяти стека. Полученные результаты были проверены на корректность сравнением с эталонными данными, что доказало возможность применения разработанных плагинов для решения реальных задач анализа. Замедление, накладываемое плагинами, значительное, однако всё ещё позволяющее работать с исследуемой системой.

В дальнейшем разработка будет сосредоточена на реализации новых инструментов анализа, базирующихся на результатах описанной в данной статье работы.

Список литературы / References

[1]. Luk C.-K.. Cohn R. et al. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. ACM SIGPLAN Notices, vo. 40, issue 6, 2005, pp 190-200.

[2]. Nethercote N., Seward J. Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation. In Proc. of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2007, pp. 89-100.

[3]. Dolan-Gavitt B., Leek T. et al. Virtuoso: Narrowing the Semantic Gap in Virtual Machine Introspection. In Proc. of the IEEE Symposium on Security and Privacy, 2011, pp. 297-312.
[4]. Henderson A., Prakash A. et al. Make It Work, Make It Right, Make It Fast: Building a Platform-Neutral Whole-System Dynamic Binary Analysis Platform. In Proc. of the International Symposium on Software Testing and Analysis. 2014, pp. 248-258.
[5]. Henderson A., Yan L.K. et al. DECAF: A Platform-Neutral Whole-System Dynamic Binary Analysis Platform. IEEE Transactions on Software Engineering, vol. 43, no. 2, 2017, pp. 164-184.
[6]. Zeng J., Fu Y., Lin Z. PEMU: A Pin Highly Compatible Out-of-VM Dynamic Binary Instrumentation Framework. In Proc. of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, 2015, pp. 147-160.
[7]. Bruening D., Duesterwald E., Amarasinghe S. Design and implementation of a dynamic optimization framework for Windows. In Proc. of the 4th ACM Workshop on Feedback-Directed and Dynamic Optimization (FDDO-4).
[8]. Song D., Brumley D. et sl. BitBlaze: A new approach to computer security via binary analysis. In Proc. of the International Conference on Information Systems Security, 2008, pp. 1-25.
[9]. Bellard F. QEMU, a Fast and Portable Dynamic Translator. In Proc. of the Annual Conference on USENIX Annual Technical Conference, 2005, pp. 41-46.
[10]. Dolan-Gavitt B., Leek T. et al. Tappan Zee (North) Bridge: Mining Memory Accesses for Introspection. In Proc. of the 2013 ACM SIGSAC Conference on Computer & Communications Security, 2013, pp. 839-850.
[11]. Python scriptable reverse engineering sandbox, a virtual machine instrumentation and inspection framework based on qemu. Available at: https://github.com/Cisco-Talos/pyrebox, accessed 24.08.2021.
[12]. Icebox. Available at: https://github.com/thalium/icebox, accessed 24.08.2021.
[13]. Volatility framework – volatile memory extraction utility frame-work. Available at: https://github.com/volatilityfoundation/volatility, accessed on 24.08.2021.
[14]. Winbagility. Available at: https://winbagility.github.io/, accessed on 24.08.2021.
[15]. Tanenbaum A.S., Bos H. Modern operating systems. 4th edition. Pearson, 2014, 1136 p.
[16]. Vasiliev I., Dovgalyuk P., Klimushenkova M. Selective Instrumentation Mechanism and its Application in a VirtualMachine. In Proc. of the Ivannikov Memorial Workshop (IVMEM), 2019, pp. 72-76.

Информация об авторах / Information about authors

Иван Александрович ВАСИЛЬЕВ – разработчик программного обеспечения. Сфера научных интересов: отладка, интроспекция и инструментирование виртуальных машин, динамический анализ бинарного кода, эмуляторы.

Ivan Aleksandrovich VASILIEV is a software developer. Research interests: debugging, introspection and instrumentation of virtual machines, dynamic analysis of binary code, emulators.

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

Pavel Mikhailovich DOVGALYUK – Senior Researcher, Associate Professor, Candidate of Technical Sciences. Research interests: introspection and instrumentation of virtual machines, dynamic code analysis, emulators.

Мария Анатольевна КЛИМУШЕНКОВА – разработчик программного обеспечения. Сфера научных интересов: отладка, интроспекция и инструментирование виртуальных машин, динамический анализ бинарного кода, эмуляторы.

Maria Anatolyevna KLIMUSHENKOVA is a software developer. Research interests: debugging, introspection and instrumentation of virtual machines, dynamic analysis of binary code, emulators.

DOI: 10.15514/ISPRAS-2021-33(6)-5

Kotlin с точки зрения разработчика статического анализатора

[1,2] *В.О. Афанасьев, ORCID: 0000-0002-8036-0633 <vafanasiev@ispras.ru>*
[1] *С.А. Поляков, ORCID: 0000-0002-8542-8035 <inly@ispras.ru>*
[1] *А.Е. Бородин, ORCID: 0000-0003-3183-9821 <alexey.borodin@ispras.ru>*
[1,3] *А.А. Белеванцев, ORCID: 0000-0003-2817-0397 <abel@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Национальный исследовательский университет Высшая школа экономики, 101000, Россия, г. Москва, ул. Мясницкая, д. 20*
[3] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*

Аннотация. В статье описывается статический анализатор для поиска ошибок и анализа метрик и отношений в программах на языке Kotlin. Анализатор был реализован с помощью расширения инструмента *Svace*, разрабатываемого в ИСП РАН. В статье описываются проблемы, с которыми мы столкнулись в ходе выполнения работы, и предложенные методы их решения, а также экспериментальные результаты полученного анализатора. Инструмент умеет не только анализировать программы на языке Kotlin, но также поддерживает анализ смешанных проектов, использующих языки Java и Kotlin. Надеемся, что статья будет полезна разработчикам статических анализаторов, а также тем, кто проектирует новые языки программирования.

Ключевые слова: статический анализ; поиск ошибок; анализ метрик; уязвимости; Kotlin; JVM; байткод

Для цитирования: Афанасьев В.О., Поляков С.А., Бородин А.Е., Белеванцев А.А. Kotlin с точки зрения разработчика статического анализатора. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 67-82. DOI: 10.15514/ISPRAS–2021–33(6)–5

Kotlin from the perspective of a static analyzer developer

[1,2] *V.O. Afanasyev, ORCID: 0000-0002-8036-0633 <vafanasiev@ispras.ru>*
[1] *Polyakov S.A., ORCID: 0000-0002-8542-8035 <inly@ispras.ru>*
[1] *A.E. Borodin, ORCID: 0000-0003-3183-9821 <alexey.borodin@ispras.ru>*
[1,3] *A.A. Belevantsev, ORCID: 0000-0003-2817-0397 <abel@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *National Research University Higher School of Economics, 20, Myasnitskaya Str., Moscow, 101000, Russian Federation*
[3] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*

Abstract. The paper describes a static analysis for finding defects and computing metrics for programs written in the Kotlin language. The analysis is implemented in the Svace static analyzer developed at ISP RAS. The paper focuses on the problems we met during implementation, the approaches we used to solve them, and the experimental results for the tool we have built. The analyzer supports not only Kotlin analysis, but is also

capable of analyzing mixed projects that use both Java and Kotlin languages. We hope that the paper might be useful to static analysis developers and language designers.

Keywords: static analysis; search for defects; vulnerabilities; Kotlin; JVM; bytecode

For citation: Afanasyev V.O., Polyakov S.A., Borodin A.E., Belevantsev A.A. Kotlin from the perspective of a static analyzer developer. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 67-82 (in Russian). DOI: 10.15514/ISPRAS–2020–33(6)–5

1. Введение

Kotlin – относительно молодой язык, разрабатываемый компанией JetBrains [1]. Язык является статически-типизированным и поддерживает парадигмы объектно-ориентированного и функционального программирования. При разработке языка особое внимание уделялось типобезопасности. Язык использует Java Virtual Machine. В мае 2017 года компания Google сообщила, что язык Kotlin будет стандартным языком для разработки ОС Android и приложений для неё наравне с языком Java [2].

Исходный код на языке Kotlin может быть скомпилирован в три различных варианта промежуточного представления: JVM-байткод [3] при использовании платформы Kotlin/JVM, код на JavaScript при использовании Kotlin/JS, LLVM-биткод при компиляции с использованием Kotlin/Native. При этом надо отметить, что один и тот же код может успешно компилироваться под одну платформу, но не под другую, если этот код использует платформо-зависимые библиотеки или API. Так, например, мобильное приложения для Android не может быть полностью собрано при помощи инструментария Kotlin/Native.

В данной работе мы рассматриваем только компиляцию для JVM. Отметим, что в данном случае код, написанный на языке Java, может вызываться из кода на языке Kotlin и наоборот.

Язык Kotlin спроектирован таким образом, чтобы исключить возможность возникновения многих ошибочных ситуаций в коде программ. В частности, для исключения ошибки «разыменование нулевого указателя» [4] система типов языка Kotlin поддерживает два вида типов: те, что могут принимать значение null (nullable references), и те, что не могут (non-nullable references). При этом разыменование значения null может произойти только в следующих случаях:

- явное небезопасное разыменование nullable-объекта при помощи операторов !!$ и *as*;
- передача объекта в какой-либо метод в процессе его конструирования до инициализации всех полей (leaking this);
- небезопасное взаимодействие с Java-кодом.

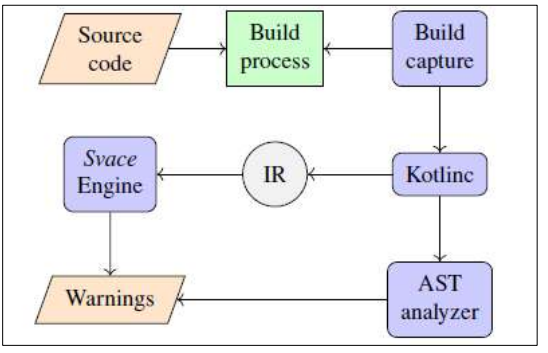


Рис. 1. Схема анализа
Fig. 1. Analysis scheme

Для языка доступно несколько видов легковесных анализаторов (линтеров): detekt [5] и ktlint [6]. Тем не менее, нам не известно о существующих статических анализаторах, выполняющих

глубокий межпроцедурный анализ. Мы решили восполнить этот пробел и добавить поддержку анализа языка Kotlin в инструмент статического анализа *Svace* [7, 8].

Среди языков, поддерживаемых инструментом *Svace*, находится язык Java. Анализатор использует байткод JVM как промежуточное представление для анализа. Поэтому реализация статического анализатора на основе байткода JVM для языка Kotlin представлялась несложной задачей. В данной статье мы опишем проблемы, с которыми столкнулись.

На рис. 1 показана схема анализа. Анализ можно разделить на два важных этапа: контролируемая сборка проекта с генерацией промежуточного представления программы и статический анализ получившегося представления. Эти этапы будут описаны в разд. 4 и 6 соответственно.

2. Перехват сборки

Преимуществом анализатора *Svace* является поддержка автоматического анализа кода: участие пользователя в нем сведено к минимуму. Для анализа проекта необходимо запустить утилиту перехвата сборки, подав на вход оригинальную команду сборки:

```
svace build make.
```

После этого запускается анализ с помощью команды:

```
svace analyze.
```

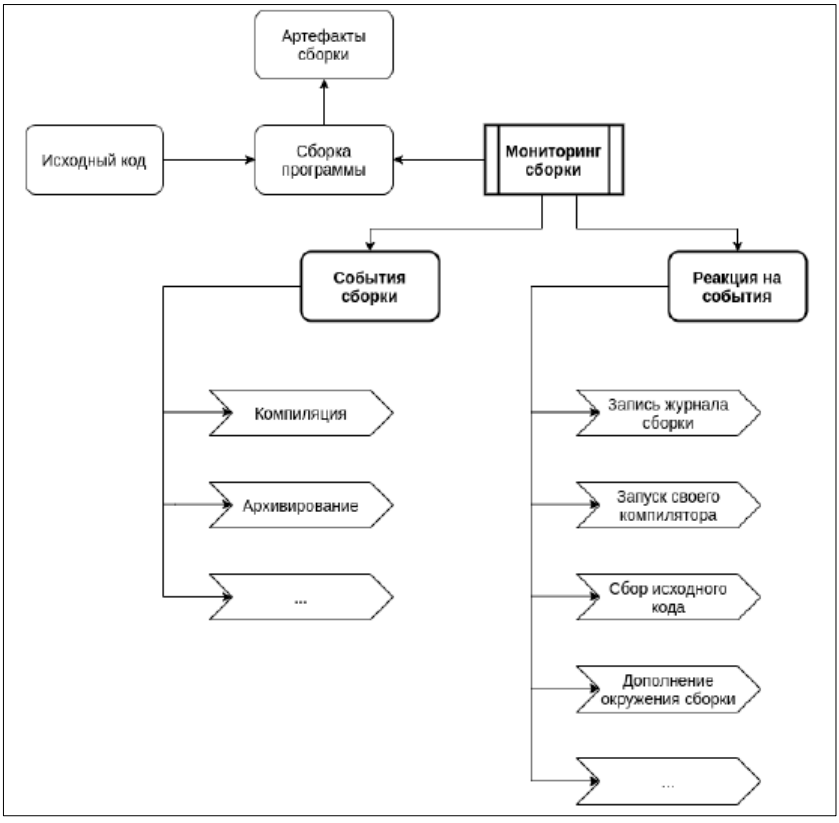


Рис. 2. Устройство контролируемой сборки
Fig. 2. Build capturing structure

Назовём *контролируемой сборкой* процесс запуска оригинальной сборки с отслеживанием интересующих нас процессов. Контролируемая сборка необходима для извлечения информации о том, как именно предполагалось компилировать тот или иной файл проекта.

В частности, для языка Kotlin необходимо правильно задать пути к JAR-библиотекам, пути к директориям с исходными файлами на языке Java, пути к плагинам компилятора, версию языка и т.п.

Svace проводит мониторинг процесса сборки без его искажения и отслеживает выполнение определенных действий – событий сборки: запуск JVM, команды компиляции и т.д. Также в процессе контролируемой сборки проводится первый этап трансляции исходного кода проекта в промежуточное представление *Svace*.

Устройство контролируемой сборки [9] в анализаторе *Svace* представлено на рис. 2.

На каждое событие сборки анализатор *Svace* реагирует специальным образом, собирая необходимую для анализа информацию. Для событий сборки проектов на языке Kotlin реализованы следующие реакции:

- добавление специального Java-агента [10] в параметры запуска виртуальной машины;
- запуск собственного компилятора Kotlin}
- сбор исходных кодов и библиотек.

Компилятор языка Kotlin представляет из себя JAR-библиотеку и вызывается через программный интерфейс. API компилятора используется такими инструментами автоматической сборки, как Gradle, Maven и Ant. Бинарный файл *kotlinc*, поставляемый вместе с компилятором, является bash-скриптом для запуска компилятора на ОС семейства Linux. Для запуска компилятора на ОС семейства Windows используется bat-скрипт *kotlinc.bat*. Оба скрипта используют API компилятора для его запуска. Таким образом, в процессе сборки любого проекта на языке Kotlin происходит не запуск компилятора в виде процесса, а вызов некоторого метода в виртуальной машине Java. Анализатор реагирует на запуск виртуальной машины Java (событие сборки) передачей виртуальной машине пути к библиотеке с Java-агентом для перехвата Kotlin-компиляций, выполняемых через программный интерфейс.

В общем случае Java-агент – это JAR-библиотека, которой виртуальная машина сообщает о загрузке какого-либо класса и позволяет изменить этот класс. В случае анализатора *Svace*, если переданный класс реализует интерфейс *org.jetbrains.kotlin.cli.common.CLITool*, то метод *exec* в этом классе будет проинструментирован таким образом, что при его вызове будет дополнительно вызван специальный метод *interceptKotlinc*, которому в качестве параметров будут переданы необходимые данные о компиляции. Данный метод запускает специальный *dummy*-процесс, параметрами которого являются все собранные данные о компиляции.

Основная сложность данной реакции заключается в том, что нет официального и задокументированного API компилятора.

Для построения промежуточного представления мы используем модифицированный компилятор Kotlin (собственный компилятор), который запускается в ответ на событие «запуск dummy-процесса». В модифицированном компиляторе отключены оптимизации, которые могут затруднить анализ, и реализовано сохранение дополнительной информации, о чём будет подробнее написано в разд. 3.

Язык Kotlin активно развивается и имеет множество версий. При этом обратная совместимость версий компилятора [11] поддерживается не в полном объеме.

На данный момент актуальной версией является версия 1.5.31.

Собственный компилятор Kotlin в *Svace* основан на версии 1.5.10. По этой причине, и поскольку пользовательский проект может использовать любую доступную версию компилятора, опции, с которыми был запущен оригинальный компилятор, необходимо переработать перед запуском собственного компилятора. Например, опция *–language-version*

1.2 запрещена для использования, начиная с компилятора версии 1.5.10. Данную опцию необходимо исключить из списка опций для запуска собственного компилятора.

Для компилятора Kotlin существует набор стандартных плагинов, а также пользователь может создавать собственные плагины. Например, стандартный плагин kapt [12] реализует процессор аннотаций. Версия плагина должна быть совместима с версией используемого компилятора. По этой причине при запуске собственного компилятора необходимо использовать собственные стандартные плагины. Таким образом, опцию *-Xplugin*, используемую для передачи путей к JAR-библиотекам с плагинами компилятора также необходимо переработать. Пути к известным стандартным плагинам должны быть заменены на пути к собственным плагинам компилятора из дистрибутива *Svace*. Если при сборке проекта используются нестандартные плагины компилятора, несовместимые с компилятором версии 1.5.10, успешный анализ таких проектов не гарантируется.

Файлы с исходным кодом используются для показа предупреждений анализа. Такие файлы могут быть удалены или перемещены в процессе сборки, поэтому сохранить их следует немедленно при обработке соответствующего события сборки. Для сохранения исходных файлов было необходимо реализовать в собственном компиляторе механизм, помечающий такие файлы.

Библиотеки в Kotlin, так же, как и в Java, принято распространять в виде JAR-библиотек. Эти библиотеки представляют собой запакованный байткод и используются анализатором для увеличения точности анализа. Используемые JAR-библиотеки также являются артефактами сборки.

3. Генерация промежуточного представления

Компиляторы Java и Kotlin на вход получают исходный код, производят синтаксический разбор и создают абстрактное синтаксическое дерево. Результатом работы компиляторов является байткод JVM.

По сравнению с Java язык Kotlin имеет значительно больше синтаксического сахара, что приводит к следующим проблемам:

- многие детали оригинальной программы теряются на уровне байткода;
- байткод содержит конструкции, которые являются результатом трансляции конструкций языка Kotlin и не были написаны непосредственно программистом.

Поясним обе эти проблемы. Анализатор *Svace* осуществляет поиск дефектов в исходном коде. Критерием выдачи предупреждения является то, что код надо поправить. При этом ошибка не обязательно будет проявляться во время выполнения. Например, это может быть бесполезное сравнение либо код в функции, которую никто не вызывает.

Так как не все свойства языка сохраняются при трансляции в байткод, не все ошибки могут быть найдены. Частичным решением этой проблемы являются детекторы на основе абстрактного синтаксического дерева (АСД). Но эти детекторы имеют свои хорошо известные ограничения. Поэтому часть дефектов, детали которых отсутствуют в байткоде и которые сложны для АСД, не может быть найдена.

Другой проблемой, связанной с трансляцией, является генерация бесполезного либо недостижимого кода. Компилятор генерирует его для множества конструкций языка Kotlin. Чтобы не выдавать бесполезные предупреждения, мы помечаем такие конструкции, как «сгенерированные компилятором». Например, из-за того, что в JVM отсутствуют булевский тип (он заменяется численным типом) и соответствующая для него инструкция отрицания, оператор отрицания в исходном коде часто раскрывается в ветвление. Отметим, что данная проблема актуальна и для Java кода [13].

На рис. 3 представлен пример такой генерации кода. Инструкции, соответствующие байтам 5, 6, 9, 10, 13 в байткоде, образуют ветвление, которое неявно присутствует в исходном коде

благодаря использованию оператора отрицания. Но поскольку ранее в условном выражении значение переменной *x* было проверено на истинность, одна из ветвей такого кода является недостижимой, о чём и сообщит статический анализатор.

```
1: fun foo(x: Boolean) {
2:     if (x) {
3:       // Unreachable code
4:       // warning is incorrect
5:       smth(!x)
6:     }
7: }
```

```
0:  iload_1
1:  ifeq 17
4:  aload_0
5:  iload_1
6:  ifne 13
9:  iconst_1
10: goto 14
13: iconst_0
14: invokevirtual smth:(Z)V
```

Рис. 3. Пример с оператором отрицания
Fig. 3. Negation operator example

```
1: var err = 0
2: try {
3: return smth()
4: } catch(e: RuntimeException) {
5: err = 1
6: } finally {
7: if (err == 1) {
8: report()
9: }
10: finish()
11: }
12: return err
```

```
0: iconst_0
1: istore_2
2: aload_0
3: invokevirtual smth:()I
6: istore_3
7: iload_2
8: iconst_1
9: if_icmpne 16
12: aload_0
13: invokevirtual report:()V
16: aload_0
17: invokevirtual finish:()V
20: iload_3
21: ireturn
22: astore_3
23: iconst_1
24: istore_2
25: iload_2
26: iconst_1
27: if_icmpne 34
30: aload_0
31: invokevirtual report:()V
34: aload_0
35: invokevirtual finish:()V
38: goto 59
41: astore 4
43: iload_2
44: iconst_1
45: if_icmpne 52
48: aload_0
49: invokevirtual report:()V
52: aload_0
53: invokevirtual finish:()V
56: aload 4
58: athrow
59: iload_2
60: ireturn
```

Рис. 4. Пример с try-catch-finally
Fig. 4. Try-catch-finally example

Чтобы избежать выдачи ложных предупреждений, подобный код был помечен при помощи специальной аннотации, которая сообщает статическом анализатору о том, что такой код сгенерирован компилятором[1].

Ещё одна проблема генерации промежуточного представления связана с трансляцией finally-блоков. Компилятор языка Kotlin при генерации конструкций try-catch-finally генерирует дублирующийся код, соответствующий коду из finally-блока, и добавляет его после кода из блока try, после кода каждого из блоков catch и перед каждой инструкцией *break, continue* и *return*, которые могут совершить выход из конструкции try-catch-finally. Дублирование finally-блоков неудобно тем, что исходный код программы перестаёт взаимнооднозначно отображаться на байткод – каждой строке исходного кода может соответствовать сразу несколько инструкций из промежуточного представления. Следовательно, сгенерированный байткод может содержать пути, которые будут недостижимы, но при этом исполнение программы может проходить через соответствующие точки исходного кода. На рис. 4 представлен пример, взятый из [13]. Если код, представленный в try-блоке, завершится без исключений, то значение переменной *err* будет равно нулю, и условное выражение из finally-блока будет бессмысленно, о чём и сообщит статический анализатор. Чтобы избежать подобных ложных срабатываний и сгенерировать код из finally-блока лишь единожды, было использовано решение, применённое ранее при модификации компилятора javac, с использованием инструкций *jsr* и *ret* [13].

Следующей проблемой в генерации промежуточного представления компилятором языка Kotlin является генерация специальных intrinsic-вызовов для проверок значений на *null*. Например, при изменении типа значения с nullable в non-nullable при помощи операторов !! и *as* генерируется intrinsic-вызов *kotlin.jvm.internal.Intrinsics.checkNotNull*. Поведение этой функции неизвестно статическому анализатору, поэтому такой вызов заменяется на вызов специальной функции, которая сообщает анализатору, что происходит разыменование, и после этого объект не может принимать значение *null*. Все другие intrinsic-вызовы, генерируемые компилятором Kotlin, не имеют какого-либо значения для статического анализа, поэтому генерация таких вызовов была отключена при помощи соответствующих опций компилятора.

В языке Kotlin в отличии от Java иногда допустима перегрузка по возвращаемому значению, если компилятор может вывести тип возвращаемого значения из контекста. Например, такое возможно из-за более умной трансляции generic-типов параметров функций. В листинге 1 приведены примеры кода, идентичного в языках Java и Kotlin, но компилятор языка Java выдаёт ошибку при компиляции такого кода, а код на языке Kotlin может быть корректно скомпилирован при помощи kotlinc.

```
1: interface Example {
2:    Integer smth(List<Integer> l);
3:    String smth(List<String> l);
4: }
```

```
1: interface Example {
2:    fun smth(l: List<Int>): Int
3:    fun smth(l: List<String>): String
4: }
```

Листинг 1. Пример с перегрузкой по возвращаемому значению
Listing 1. Return value overload example

Такие перегрузки возможны благодаря тому, что в JVM в сигнатуру метода входит и тип возвращаемого значения метода, из-за чего методы с одинаковыми именами и типами параметров могут быть различены по типу возвращаемого значения. Компилятор языка Kotlin частично использует эту возможность JVM.

[1] Фактически, весь код генерируется компилятором. В данном случае инструкция ветвления появляется только в байткоде, а в исходном коде нет соответствующих инструкций ветвления.

Для поддержки такой возможности в анализаторе *Svace* пришлось исправить вид сигнатуры методов таким образом, чтобы в нём присутствовал и тип возвращаемого значения.

Существенным отличием JVM-байткода, сгенерированного компилятором Kotlin, от байткода, сгенерированного компилятором Java, является наличие большого числа синтетических функций, которые не представлены явно в исходном коде. К примеру, одним из нововведений языка Kotlin являются функции с параметрами по умолчанию, которые отсутствуют в языке Java. Для поддержки таких функций в байткоде генерируется специальная функция с суффиксом *$default*, в которую, помимо аргументов исходной функции, передаются как минимум два дополнительных аргумента, по которым вычисляется, какой из параметров принимает значение по умолчанию.

Так как такие синтетические функции скрываются при отладке, для них генерируются очень мало отладочной информации – например, зачастую отсутствует такая информация, как названия, типы и индексы локальных переменных и параметров. Наличие подобного рода функций существенно ухудшает понятность и полезность предупреждений, выдаваемых статическим анализатором. Чтобы улучшить качество выдаваемых предупреждений, компилятор kotlinc был модифицирован таким образом, чтобы для функций генерировалось больше отладочной информации: для многих выражений, в частности многострочных, была улучшена генерация атрибута *LineNumberTable*, хранящего взаимное соответствие строк исходного кода с инструкциями байткода; все локальные переменные и параметры, в том числе синтетические, добавляются в атрибут *LocalVariableTable* с корректными именами, индексами и типами. Тем не менее, наличие подобных синтетических функций всё ещё может создавать некоторые проблемы, поэтому в будущем нам представляется возможным изменение генерации промежуточного представления с целью полного или частичного отключения генерации этих функций.

Значительную сложность добавило наличие в данном языке встраиваемых (inline) функций, которые очень часто используются; в частности, большое количество функций стандартной библиотеки являются встраиваемыми. В отличие от языка C++, где наличие ключевого слова *inline* является только подсказкой для компилятора и может быть проигнорировано, язык Kotlin не позволяет полностью отключить такое поведение. Рассмотрим пример, представленный в листинге 2. Оператор *return*, вложенный в лямбда-выражение, передаваемое в функцию *map*, относится к внешней функции *sumOrNull*. Если в списке встретится строка, не являющаяся целым числом, то функция завершится и вернёт *null*. Такие *return*, которые находятся внутри лямбда-выражения, но завершают внешнюю функцию, называются нелокальными (non-local). Заметим, что функция *map* является встраиваемой. Если бы функция *map* и передаваемое внутрь неё лямбда-выражение не были встроены в место вызова, то такое поведение оператора *return* было бы невозможным, поскольку компилятор языка Kotlin не поддерживает нелокальные *return* внутри лямбда-выражений, передаваемых в обычные, не встраиваемые функции.

```
1: fun sumOrNull(strings: List<String>): Int? {
2:    return strings.map {
3:       val x = it.toIntOrNull()
4:       if (x == null) return null
5:    x
6:  }.sum()
7: }
```

Листинг 2. Пример с встраиваемыми функциями
Listing 2. Inline functions example

Наличие встраиваемых функций в коде существенно влияет на качество статического анализа. Так как зачастую встраивание функций создаёт код, являющийся недостижимым или излишним, то использование таких функций в анализируемом коде будет создавать большое количество ложных предупреждений. Листинг 3 иллюстрирует такое ложное срабатывание. Используемая функция *substring* принимает non-nullable параметры, поэтому

в месте вызова неявно генерируется проверка передаваемых аргументов на *null.substring* – это функция-расширение, поэтому в качестве её аргументов также передаётся объект, на котором вызывается эта функция. Так как функция вызывается два раза на одном и том же объекте, то его проверка на значение *null* будет осуществлена дважды. Следовательно, вторая проверка будет излишней, о чём и сообщит статический анализатор. В данном случае количество выданных ложных предупреждений было уменьшено благодаря изменениям в генерации intrinsic-вызовов, которые были описаны выше.

```
1: fun String.duplicateBefore(position: Int): String {
2:   return substring(0, position)
3:     + substring(0, position)
4: }
```

Листинг 3. Пример с встраиваемыми функциями из стандартной библиотеки
Listing 3. Inline functions from standard library example

В некоторых случаях наличие встраиваемых функций в коде может, наоборот, улучшать результаты анализа, так как становится проще понимать эффект применения таких функций в месте вызова. Например, стандартная библиотека языка Kotlin имеет большое количество функций высшего порядка, которые позволяют работать с коллекциями. Поскольку в инструменте *Svace* не реализовано моделирование таких функций, то при отключённом встраивании таких функций было бы сложнее понять, какой эффект они оказывают на результирующую коллекцию.

Также нужно отметить, что даже частичное отключение встраиваемых функций может существенно повлиять на время и результаты анализа. Отключение встраивания для функций означало бы, что для каждого лямбда-выражения, используемого при вызове, генерировался бы анонимный класс с виртуальным методом *invoke*, позволяющим выполнить данное лямбда-выражение. А из-за того, что использование встраиваемых функций в Kotlin является широко распространённой практикой, число анонимных классов и виртуальных функций, генерируемых компилятором, существенно возрастёт.

4. Анализ

Инструмент *Svace* использует анализ на основе резюме. На вход движку анализа подаются модули, представляющие из себя модифицированный байткод. Анализатор читает эти модули, строит граф вызовов, и начинает обход функций поочерёдно начиная с листьев графа вызовов. Вызываемые функции посещаются до вызывающих.

Каждая функция анализируется только один раз. После анализа создаётся резюме, которое описывает поведение функции, интересное для анализатора. При анализе инструкции вызова функции используется только её резюме, которое транслируется в контекст вызова.

Анализ отдельной функции является потоково-чувствительным. Анализ отличает различные поля структур и отдельные элементы массивов. Кроме этого, анализ имеет чувствительность к путям, то есть способен отличать отдельные пути, проходящие через граф потока управления. Для определения невыполнимых путей используется SMT-решатель. Подробнее про анализ отдельной функции можно прочитать в [14].

Мы реализовали детекторы для следующих видов ошибок:

- утечки ресурсов;
- использование ресурса после освобождения;
- разыменования нулевых указателей;
- недостижимый код;
- деление на ноль;
- отсутствие проверки кода возврата библиотечных функций;
- переполнение буфера данными из внешних источников.

5. Совместный анализ Kotlin и Java

Результат компиляции исходного кода Kotlin в байткод можно запаковать в JAR-библиотеку и использовать её методы в проекте, написанном на Java, передавая компилятору путь к библиотеке.

Кроме этого, разработчики языка Kotlin реализовали возможность вызывать методы, исходный код которых написан на Java, причём не только с помощью передачи пути до соответствующей JAR-библиотеки, но и с помощью передачи путей до исходного Java-кода, где эти методы определены. Таким образом, существует возможность разрабатывать проекты, в которых используются Kotlin и Java одновременно, более того, в которых существует циклическая зависимость по коду.

Ошибка в проекте может проявляться на пути, проходящим через Java и Kotlin код. Листинг [4] содержит пример такой ошибки. Метод *KotlinPartKt.test* вызывает метод *JavaPart.getBuggyString*, который вернет *null*, поскольку в качестве фактического параметра передана строка *bug*. Далее результат вызова метода разыменовывается, что приведет к *NullPointerException*.

```
// file: src/main/kotlin/kotlinPart.kt
1: package svace.test
2:
3: fun test(): Unit {
4:    val s: String = JavaPart.getBuggyString("bug")
5:    println(s.substring(s.lastIndexOf('/')))
6: }
7:
8: fun getBuggyString(b: String): String? {
9:    return if (b == "bug") null else "$b␣is␣OK"
10: }
...
// file: src/main/java/svace/test/JavaPart.java
1: package svace.test;
2:
3: public class JavaPart {
4:    private void test() {
5:       String s = KotlinPartKt.getBuggyString("bug");
6:       System.out.println(s.substring(s.lastIndexOf('/')));
7:    }
8:
9:    public static String getBuggyString(String b) {
10:      if (b.equals("bug")) return null;
11:      return b + "␣is␣OK";
12:   }
13: }
```

Листинг 4. Пример, для анализа которого необходим совместный анализа Kotlin и Java
Listing 4. Cross language analysis example

Заметим, что в определении метода *JavaPart.getBuggyString* возвращаемое значение не имеет аннотации *Nullable*. Приведенный пример также демонстрирует один из случаев, когда, несмотря на null-безопасную систему типов языка, в программе на Kotlin произойдёт разыменование нулевого указателя. В Java-части проекта метод *JavaPart.test* вызывает метод **KotlinPartKt.getBuggyString**, возвращающий *null*, который будет затем разыменован.

Соответственно, для поиска таких ошибок от анализатора требуется анализ кода для обоих языков с учётом зависимостей.

В анализаторе *Svace* для этого был разработан специальный режим работы, в котором строится общий граф вызовов для целого проекта и учитываются зависимости между Kotlin и Java кодом. Затем на общем графе вызовов проводится полноценный анализ.

Анализ для приведённого листинга 4 выдаёт следующие предупреждения:

1.
   - Pointer 's' returned from function 'KotlinPartKt.getBuggyString' at JavaPart.java:5 may be null, and it is dereferenced at JavaPart.java:6.
   - Variable 's' is dereferenced at JavaPart.java:6
   - Null assign at kotlinPart.kt:9

2.
   - Pointer 's' returned from function 'JavaPart.getBuggyString' at kotlinPart.kt:4 may be null, and it is dereferenced at kotlinPart.kt:5.
   - Variable 's' is dereferenced at kotlinPart.kt:5
   - Assign null at JavaPart.java:10

6. Анализ на основе абстрактного синтаксического дерева

Компилятор Kotlin предоставляет АСД, информацию о типах переменных, иерархии классов, а также константах времени компиляции. Более того, компилятор имеет встроенный анализатор АСД. Подобные анализаторы позволяют находить опечатки в исходном коде. Нами были реализованы детекторы для обнаружения следующих дефектов:

- сравнение вместо присваивания, то есть использование оператора равенства, не влияющего на выполнение программы;
- повторяющиеся условия, то есть дублирование условий в операторе с несколькими условиями;
- некорректные границы интервала: при создании интервала вида *a..b*, такого, что $a > b$, в Kotlin создаётся пустой интервал
- вызов метода *next()* в реализации метода *hasNext()* в классе, реализующем интерфейс *Iterator*.

7. Спецификации

Спецификации представляют собой код на анализируемом языке, который добавляется в анализатор. В спецификациях используются вызовы специальных функций, которые имеют особое значение для анализатора. Функции из спецификаций анализируются, и далее их резюме используется вместо резюме оригинальных функций проекта. Спецификации позволяют решить две проблемы:

- добавить семантику для библиотечных функций, код которых отсутствует;
- сообщить анализатору детали поведения функции, которые не могут быть выведены средствами статического анализа.

В листинге 5 представлен пример спецификации, а также код, дефект в котором анализатор обнаружит только при наличии соответствующей спецификации. Рассмотрим пример подробнее. В методе *testTainted* значение *num* вычисляется с помощью вызова метода *toInt* объекта типа *String*. Затем значение num используется в качестве индекса доступа к массиву.

Как правило, функции преобразования строк в число используются для данных из внешних источников. Мы используем эвристику, заключающуюся в том, что результат таких функций необходимо проверить. Это позволяет упростить анализ и не проверять, что строка получена из внешнего источника. Использование целых чисел из внешнего источника как индекс доступа к массиву может привести к возникновению исключения. Если эти данные

действительно из внешних источников, то злоумышленник сможет контролировать такое поведение. В отличие от языка C, здесь это не является уязвимостью, но всё ещё остаётся ошибкой в программе.

```
// specification source file
1: package kotlin.text
2:
3: import ru.isp.svace.sf.*;
4:
5: public fun String.toInt(): Int {
6:    val res = SpecFunc.sf_get_some_int() as Int
7:    SpecFunc.sf_set_tainted_int(res)
8:    return res
9: }
...
// test source file
1: fun testTainted(number: String) {
2:    val num: Int = number.toInt()
3:    val x: IntArray = intArrayOf(1, 2, 3)
4:    print(x[num])
5: }
```

Листинг 5. Пример спецификации
Listing 5. Specification example

Функция *SpecFunc.sf_set_tainted_int* сообщает анализатору, что её параметр получен из внешнего источника и требует проверки. Далее это свойство распространяется анализатором. И на 4 строке будет выдано сообщение об ошибке, так как индекс доступа к массиву может лежать за границами, заданными его размером.

Помимо спецификаций, добавленных разработчиками анализатора, пользователь может добавлять свои собственные спецификации для каждого анализируемого проекта.

8. Анализ метрик и отношений

В инструменте *Svace* содержится отдельный компонент, служащий для определения сущностей программы, их метрик, связей между ними. Сущностями обычно являются методы, глобальные переменные, поля, классы, файлы и каталоги; связями --- случаи чтения или записи одной сущности другой, вызова, включения, наследования и пр.; метриками --- количественные характеристики сущностей или групп связей. Более подробно об устройстве такого анализа для языков C/C++ можно прочесть в статье [15].

Для языка Java подсчет метрик и отношений ведется схожим с поиском дефектов способом: сначала выполняется контролируемая сборка для запуска собственного компилятора Java, который вычисляет часть метрик, могущих быть определенными только в момент разбора программы с полным доступом к исходному коду; затем запускается анализ, который считывает Java-байткод и специальные аннотации в нем, содержащие вычисленные компилятором метрики, и выполняет окончательный анализ, как агрегируя уже посчитанные метрики для всей программы, так и считая часть метрик по байткоду.

Учитывая, что язык Kotlin компилируется в байткод Java, было принято решение повторно использовать имеющийся анализатор Java-байткода и доработать собственный компилятор Kotlin для подсчета тех же метрик, что для Java также вычисляются в компиляторе. Задача такого подсчета внутри компилятора имеет инженерный характер. Отметим две интересные особенности. Во-первых, в отличие от компилятора javac абстрактные синтаксические деревья компилятора Kotlin полностью сохраняют всю информацию о лексемах, которые были использованы при построении данного дерева, и тем самым модификация лексического анализа не требуется: нужные данные можно получить на поздних этапах компиляции. Во-вторых, при обновлении компилятора Kotlin до версии 1.5 оказалось, что кодогенерация для

старых версий языка (1.4 и ниже) выполняется в компиляторе из одного вида внутреннего представления (АСД-деревьев), а кодогенерация для версии 1.5 выполняется из полностью другого вида деревьев. Такое решение разработчиков Kotlin можно охарактеризовать как до некоторой степени странное; для нас это означало необходимость рефакторинга кода записи аннотаций с метриками, чтобы его можно было вызывать из всех компонентов кодогенерации для обеих версий языка.

В целом, как и в случае поиска дефектов, удалось успешно использовать анализатор байткода Java для вычисления метрик также и для Kotlin; теперь возможен и совместный анализ программ на Java и Kotlin, например, вызовы между Java и Kotlin частями успешно распознаются. Компиляторная часть анализа метрик нуждается в дальнейшей доработке для полноценной поддержки случаев генерации синтаксического сахара и исключения сгенерированного компилятором кода наподобие того, что уже было описано для случая поиска дефектов.

9. Результаты

Для оценки качества и производительности разработанного анализатора был выбран проект компилятора Kotlin [16]. Выбор данного проекта обусловлен несколькими причинами. Во-первых, это самый крупный проект с исходным кодом на языке Kotlin. Проект содержит 2423 тысячи строк Kotlin кода и 1093 тысячи строк Java кода. Во-вторых, в проекте одновременно используется и Kotlin, и Java, следовательно, мы можем протестировать анализатор в режиме совместного анализа двух языков. Наконец, мы полагаем, что в проекте, над которым работают разработчики языка, присутствует наибольшее разнообразие языковых конструкций и минимальное количество дефектов, что является вызовом для статического анализатора, нацеленного на выдачу минимального числа ложных предупреждений в процентном соотношении от общего числа предупреждений.

Оригинальная сборка проекта длится в среднем 17 минут[2], контролируемая сборка длится 87 минут. Таким образом, оригинальная сборка замедляется приблизительно в 5 раз в случае контролируемой сборки для последующего проведения анализа с помощью *Svace*. Время анализа проекта в режиме совместного анализа Kotlin и Java кода составляет 19 минут.

Оценка качества анализа – нетривиальная задача. Разметка предупреждений, выданных анализатором, – достаточно трудоёмкий процесс. Более того, анализатор находится в фазе активной разработки, и множество выдаваемых предупреждений постоянно изменяется. На данный момент размечена лишь часть актуальных предупреждений на проекте (41% от общего числа предупреждений), хотя в общей сложности было размечено более тысячи предупреждений, большая часть которых в настоящий момент не выдается в результате проделанной работы по подавлению ложных предупреждений.

Табл. 1. Оценка качества результатов анализа
Table 1. Quality evaluation of analysis results

Группа детекторов	Истинные предупреждения, %
Разыменование нуля	30
Утечка ресурсов	43
Целочисленное переполнение	89
Недостижимый код	44

Несмотря на то, что наша команда дорабатывала некоторые модули компилятора, код большой части проекта нами мало изучен. По этой причине мы не утверждаем, что все

[2] Данный и последующие замеры времени проводились на вычислительной машине с характеристиками: 8 cores 2.4GHz, 64RAM. Среднее значение – это среднее арифметическое в серии из нескольких замеров.

предупреждения размечены корректно, в особенности предупреждения, выданные межпроцедурными детекторами.

Статистика для некоторых групп детекторов приведена в табл. 1. Для оценки качества группы детекторов будем вычислять процент истинных предупреждений от общего числа размеченных предупреждений в данной группе.

Полученные результаты являются неудовлетворительными для нас в данный момент – целевое минимальное значение процента истинных предупреждений составляет 70%. Мы будем продолжать работы для достижения приемлемого качества. Но поскольку выбранный проект – это компилятор, в котором используются сложные конструкции, и над ним работают квалифицированные разработчики, такой результат мы оцениваем, как адекватный на данном этапе разработки анализатора.

Среди причин выдачи большого процента ложных предупреждений можно выделить значительный объём синтаксического сахара в исходном коде. Во многих случаях, описанных в разд. 3, мы смогли решить проблемы, возникающие в процессе генерации кода для таких конструкций. Однако работы в этом направлений ещё не закончены. Следующей причиной, на наш взгляд, является обширная стандартная библиотека Kotlin, которую еще предстоит описать с помощью механизма спецификаций, описанного в разд. 7. Последней известной нам причиной является тот факт, что в Kotlin-проектах активно используется парадигма функционального программирования. Для высокого качества анализа таких проектов от статического анализатора требуется построение графа вызовов с учётом девиртуализации. В *Svace* используются базовые алгоритмы девиртуализации, которые мы планируем совершенствовать в будущем.

10. Заключение

В статье была описана реализация анализа программ на языке Kotlin с помощью статического анализатора *Svace*. Основным принципом поддержки анализа Kotlin являлся анализ JVM-байткода, генерируемого компилятором Kotlin, так как имеющийся анализатор уже содержал поддержку анализа байткода для языка Java.

Процесс адаптации инструмента для анализа потребовал поддержки контролируемой сборки через перехват интерфейсов компиляции Kotlin, доработки компилятора Kotlin для построения адекватного для статического анализа внутреннего представления, а также доработки существующих детекторов для языка Java и создания некоторых новых детекторов, в том числе детекторов для анализа АСД-уровня. Полученные результаты приемлемы с учетом высокого качества анализируемого тестового кода, но требуются дальнейшие работы для достижения стандартного для *Svace* уровня в 70% истинных срабатываний.

Список литературы / References

[1] JetBrains s.r.o. URL: https://www.jetbrains.com, accessed October 6, 2021.
[2] P. Miller. Google is adding Kotlin as an official programming language for Android development. URL: https://www.theverge.com/2017/5/17/15654988/google-jet-brains-kotlin-programming-language-android-development-io-2017, accessed October 6, 2021.
[3] The Java Virtual Machine Specification. Java SE 8 Edition. URL: https://docs.oracle.com/javase/specs/jvms/se8/html, accessed October 6, 2021.
[4] C.A.R. Hoare. Null references: the billion dollar mistake. Presentation at QCon, 2009-08-25. URL: https://www. infoq.com/presentations/null-references-the-billion-dollar-mistake-tony-hoare, accessed October 6, 2021.
[5] Detekt analyzer. URL: https://detekt.github.io/detekt, accessed October 6, 202.
[6] An anti-bikeshedding Kotlin linter with built-in formatter. URL: https://ktlint.github.io, accessed October 6, 2021.
[7] В.П. Иванников, А.А. Белеванцев и др. Статический анализатор Svace для поиска дефектов в

исходном коде программ. Труды ИСП РАН, том 26, вып. 1, 2014 г., стр. 231-250 / V.P. Ivannikov, A.A. Belevantsev et al. Static analyzer Svace for finding of defects in program source code. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 1, 2014, pp. 231-250 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-7.

[8] А. Е. Бородин, А. А. Белеванцев. Статический анализатор Svace как коллекция анализаторов разных уровней сложности. Труды ИСП РАН, том 27, вып. 6, 2015 г., стр. 111-134 / A.E. Borodin, A.A. Belevantsev. A static analysis tool Svace as a collection of analyzers with various complexity levels. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 6, 2015, pp. 111-134 (in Russian). DOI: 10.15514/ISPRAS-2015-27(6)-8.

[9] А.А. Белеванцев, А.О. Избышев, Д.М. Журихин. Организация контролируемой сборки в статическом анализаторе svace. Системный администратор, вып. 7-8, 2017 г., стр. 135-139 / A.A. Belevantsev, A.O. Izbyshev, D.M. Zhurikhin. Monitoring program builds for Svace static analyzer. System Administrator, issues 7-8, 2017, pp/ 135-139 (in Russian).

[10] Package java.lang.instrument. URL: https://docs.oracle.com/javase/7/docs/api/java/lang/instrument/package-summary.html, accessed October 6, 2021.

[11] Kotlin Evolution. URL: https://kotlinlang.org/docs/kotlin-evolution.html, accessed October 6, 2021.

[12] Using kapt. URL: https://kotlinlang.org/docs/kapt.html, accessed October 6, 2021.

[13] А.П. Меркулов, С.А. Поляков, А.А. Белеванцев. Анализ программ на языке Java в инструменте Svace. Труды ИСП РАН, том 29, вып. 3, 2017 г., стр. 57-74 / A.P. Merkulov, S.A. Polyakov, A.A. Belevantsev. Supporting Java programming in the Svace static analyzer. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 3, 2017. pp. 57-74 (in Russian). DOI: 10.15514/ISPRAS-2017-29(3)-5.

[14] А.Е. Бородин, И.А. Дудина. Внутрипроцедурный анализ для поиска ошибок на основе символьного выполнения. Труды ИСП РАН, том 32, вып. 6, 2020 г., стр. 87-100 / A.E. Borodin, I.A. Dudina. Symbolic Execution Based Intra-Procedural Analysis for Search for Defects. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 6, 2020, pp. 87-100 (in Russian). DOI: 10.15514/ISPRAS-2020-32(6)-7

[15] А.А. Белеванцев, Е.А. Велесевич. Анализ сущностей программ на языках Си/Си++ и связей между ними для понимания программ. Труды ИСП РАН, том 27, вып. 2, 2015 г., стр. 53-64 / A.A. Belevantsev, E.A. Velesevich. Analyzing C/C++ Code Entities and Relations for Program Understanding. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 2, 2015, pp. 53-64 (in Russian). DOI: 10.15514/ISPRAS-2015-27(2)-4.

[16] Kotlin compiler project. URL: https://github.com/JetBrains/kotlin, accessed October 19, 2021.

Информация об авторах / Information about authors

Виталий Олегович АФАНАСЬЕВ – студент бакалавриата факультета компьютерных наук НИУ ВШЭ, сотрудник ИСП РАН. Сфера научных интересов: компиляторные технологии, статический анализ, JVM языки.

Vitaly Olegovich AFANASYEV – undergraduate student at the Faculty of Computer Science, NRU HSE, employee of ISP RAS. Research interests: compiler technologies, static analysis, JVM languages.

Сергей Андреевич ПОЛЯКОВ – младший научный сотрудник. Сфера научных интересов: статический анализ, параллелизм, JVM языки.

Sergey Andreevich POLYAKOV – researcher. Research interests: static analysis, concurrency, JVM languages.

Алексей Евгеньевич БОРОДИН – кандидат физико-математических наук, старший научный сотрудник. Сфера научных интересов: статический анализ исходного кода программ для поиска ошибок.

Alexey Evgenevich BORODIN – PhD, researcher. Research interests: static analysis for finding errors in source code.

Андрей Андреевич БЕЛЕВАНЦЕВ – доктор физико-математических наук, ведущий научный сотрудник ИСП РАН, профессор МГУ. Сфера научных интересов: статический анализ программ, оптимизация программ, параллельное программирование.

Andrey Andreevich BELEVANTSEV – Dr.Sc., Leading Researcher at ISP RAS, Professor at MSU. Research interests: static analysis, program optimization, parallel programming.

DOI: 10.15514/ISPRAS-2021-33(6)-6

Автоматическое исправление дефектов кода в системе Svace

С.В. Сыромятников, ORCID: 0000-0001-8753-6315 <syrom@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В данной статье рассматривается задача автоматического исправления дефектов кода на языках C/C++, найденных статическим анализатором на больших программных проектах. Описан опыт реализации соответствующего инструмента для статического анализатора Svace и принципы исправления дефектов разных типов. Особое внимание уделено исправлению разыменования нулевого указателя как наиболее важного и сложного из поддержанных в инструменте типов дефектов; приведена статистика работы инструмента по исправлению дефектов этого типа. Исследуются общие ограничения и специфика поставленной задачи, объясняется невозможность использования для её решения существующих систем автоматического исправления дефектов. В заключение кратко излагаются соображения по дальнейшему развитию реализованного продукта.

Ключевые слова: автоматическое исправление программ; статический анализ; дефект исходного кода; разыменование нулевого указателя.

Для цитирования: Сыромятников С.В. Автоматическое исправление дефектов кода в системе Svace. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 83-94. DOI: 10.15514/ISPRAS–2021–33(6)–6

Automatic Repair of Code Defects in the Svace System

S.V. Syromiatnikov, ORCID: 0000-0001-8753-6315 <syrom@ispras.ru>
Ivannikov Institute for System Programming of the RAS,
25, Alexander Solzhenitsyn Str., Moscow, 109004, Russia

Abstract. The main task described in this article is automatic fixing defects in C/C++ code found by a static analyzer on big software projects. We describe how we solved this task for Svace static analyzer and discuss main principles of automatic fixing defects of various types. We pay special attention to fixing null pointer dereference since it is the most important and sophisticated defect type among those we've supported. Statistics on fixes proposed for defects of this type is also provided. We discuss common limitations and other specificity of our task and explain why we cannot use existing automatic fixing tools for solving it. At the end we outline further steps of development of our tool.

Keywords: automatic program repair; automatic defect fixing; static analysis; code defects; null pointer dereference.

For citation: Syromiatnikov S.V. Automatic Repair of Code Defects in the Svace System. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 83-94 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–6

1. Введение

Существующие системы статического анализа позволяют находить тысячи дефектов на проектах размером в миллионы строк кода. Разработчики этих систем постоянно работают над повышением точности анализа, стремясь минимизировать количество ложных

срабатываний детекторов дефектов. Однако исправление найденных дефектов кода по-прежнему требует значительных усилий и времени разработчика. Поэтому представляется актуальной следующая

Задача: предложить пользователю корректные варианты исправления дефектов, найденных статическим анализатором на больших проектах, и/или автоматически их исправить.

Такая постановка задачи накладывает важные ограничения.

a. Большие проекты нередко не имеют исчерпывающего покрытия тестами.
b. Инструмент автоматического исправления дефектов должен иметь высокую производительность.
c. Поскольку мы разрабатываем инструмент для существующего активно используемого анализатора и для дефектов, которые он уже находит, мы крайне ограничены в модификации самого инструмента анализа. В то же время многие описанные в литературе системы основаны именно на глубокой интеграции детектора и исправителя дефектов ([1], [2]).

В силу ограничений a. и и b. для решения поставленной задачи мы не можем использовать технологии, основанные на генерации и оценке в той или иной степени случайных вариантов, например, генетическое программирование ([3], [4]).

В своих исследованиях мы ограничимся языками C/C++ как одними из самых востребованных на рынке и в то же время сложных для анализа.

Здесь и далее будем называть вариант исправления *корректным*, если полученный после его применения код удовлетворяет следующим условиям:

- синтаксически корректен;
- не содержит исходного дефекта;
- сохраняет логику программы (за исключением исходного дефекта);
- не содержит новых привнесенных дефектов;
- приемлемо отформатирован.

Отметим, что статические анализаторы в принципе не гарантируют истинность всех найденных дефектов, а ложное срабатывание детектора исправлять, разумеется, не надо. Однако проблемы оценки истинности дефекта мы в данной статье касаться не будем.

Кроме того, гарантировать корректность предлагаемых вариантов исправления не всегда возможно. Рассмотрим

Пример 1. Дефект “локальная переменная не инициализирована”.

Очевидный способ исправления данного дефекта - добавить инициализацию в определение переменной. Но каким значением ее инициализировать? Наиболее вероятное значение для численных переменных – ноль, для указателей – нулевой указатель. Однако в некоторых случаях логика программы может подразумевать другое значение, например, вместо инициализации указателя `char*` нулём `char *str = NULL;` на самом деле, должна быть его инициализация пустой строкой: `char *str = "";` Выявить формальные признаки подобной ситуации непросто. Однако вариант с инициализацией нулём следует предложить пользователю, так как с большой вероятностью он его устроит.

Назовём тип дефекта *исправимым*, если для него хотя бы в некоторых формально определённых ситуациях возможна автоматическая генерация вариантов исправления, в большинстве случаев оказывающихся корректными.

Далее мы обсудим некоторые общие аспекты поставленной задачи, опишем опыт её решения для статического анализатора Svace и подробно рассмотрим схему автоматического исправления для некоторых исправимых типов дефектов.

2. Автоматическое исправление в Svace

2.1 Общая схема

В архитектуре Svace ([5], [6]) изначально не была заложена возможность автоматического исправления. Более того, статический анализатор Svace работает над внутренним представлением, которое весьма далеко от исходного кода и из которого можно извлечь довольно ограниченный объем информации, нужной для исправления этого кода. Поэтому автоматический исправитель разрабатывался нами как отдельный инструмент, а его работа в общих чертах может быть описана схемой, изображённой на рис. 1. Отметим, что выдаваемой анализатором информации о дефекте в ряде случаев оказалось недостаточно для его исправления, но добавление в выдачу дополнительной информации гораздо проще интеграции в анализатор автоматического исправления.

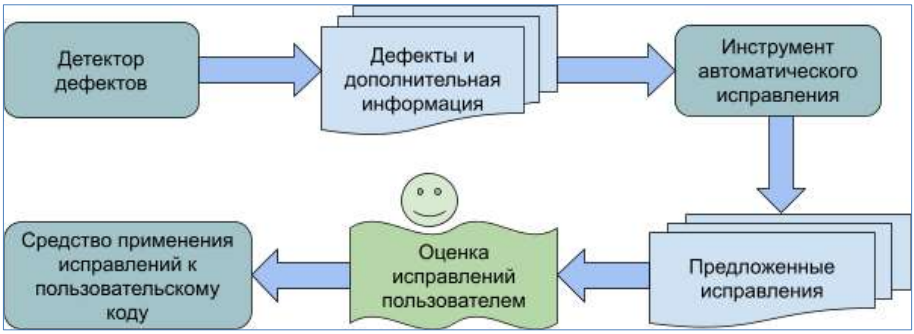


Рис. 1. Общая схема автоматического исправителя
Fig. 1. General scheme of the automatic repairer

Этап пользовательской верификации необходим по следующим причинам.

- На вход инструмента могут попасть ложные дефекты, которые исправлять не нужно.
- Инструмент может работать некорректно. Масштабное исправление кода без верификации может привести к катастрофическим последствиям.
- Как показано выше, даже в случае адекватной работы инструмента предлагаемое исправление может нарушать логику программы.
- Иногда бывает сложно выбрать из нескольких вариантов исправления оптимальный, и разумно предоставить этот выбор пользователю.

Порой бывает целесообразно предлагать автоматическое исправление не для всех дефектов данного типа, а для некоторого достаточно широкого их подмножества, осуществляя, таким образом, *предварительную фильтрацию* найденных дефектов. Важно, чтобы критерии этой фильтрации были формальными и понятными пользователю.

2.2 Входные данные

В системе Svace поддерживается автоматическое исправление для некоторых типов дефектов в коде на C/C++. Инструмент автоисправления написан на C++ на основе предоставляемой в clang технологии *LibTooling*. На вход он получает список дефектов, про каждый из которых известно следующее:

- тип;
- позиция в коде (строка и иногда столбец); для описания некоторых дефектов требуется несколько позиций, соответствующие примеры будут приведены ниже;
- указание, где искать исходный файл, который содержит ошибочный код (см. ниже);
- другая информация в зависимости от типа дефекта.

Исходные файлы, с которыми работает инструмент исправления дефектов – это частично препроцессированный пользовательский код: в нём есть все препроцессорные директивы, кроме директивы *#include*, вместо которой присутствует содержимое соответствующих заголовочных файлов; макросы также не раскрыты. Благодаря *#line*-директивам есть возможность идентифицировать фрагменты кода, заданные позициями в пользовательских файлах.

Данные файлы генерируются в процессе работы основного анализатора Svace. Их использование позволяет генерировать исправления без доступа к пользовательскому коду, например, на удалённом сервере.

2.3 Принцип работы

Стандартными средствами clang'а наш инструмент исправления строит AST для частично препроцессированного файла и находит в нем синтаксические конструкции, соответствующие нужному дефекту. Они ищутся с использованием технологии *ASTMatcher* и с учетом их известной позиции.

Пример 2. Есть дефект "Условие if-выражения всегда ложно" в позиции `foo.c:2:4`.

Чтобы найти локализацию этого дефекта в коде, нужно найти условное выражение (*IfStmt*), перейти к его условию и проверить, что это условие имеет искомую позицию.

Предлагаемое исправление дефекта – это информация о том, как надо модифицировать текст пользовательского файла, чтобы дефект пропал. Реализованное нами автоисправление основано именно на модификации текста, а не на трансформации AST'а.

Для каждого типа дефектов используется свой набор ASTMatcher-запросов, свои правила и эвристики. Во многих случаях фильтрация исправляемых дефектов происходит именно за счёт ограниченности этого набора. Иногда осуществить нужную фильтрацию узлов AST'а сложно, и рациональнее отсеять неисправимые дефекты непосредственно в анализаторе, модифицировав соответствующий детектор.

Пример 3. Дефект "память, выделенная при помощи `new[]`, освобождена при помощи `delete`".

```
 1: class Complex {
 2: public:
 3:   …
 4:   static void* operator new (size_t size) {
 5:     return new char[size];
 6:   }
 7: };
 8:
 9: void foo() {
10:   Complex *complex = new Complex();
11:   …
12:   delete complex;
13: }
```

Анализатор в состоянии отследить, что изначально память была выделена с помощью `new[]`. Стандартный способ исправления данного дефекта – замена `delete` на `delete[]` – как универсальное решение, очевидно, не подходит. Отсечь данную ситуацию (`new[]` внутри `new`) при анализе AST'а сложно, так для её обнаружения нужен анализ графа вызовов. Однако в анализаторе Svace её несложно обнаружить в процессе работы детектора.

2.4 Выходные данные

На выходе инструмент генерирует файлы в формате *unified diff* или *yaml*, описывающие необходимую для исправления дефектов модификацию кода. Применение diff-файлов к пользовательскому коду осуществляется стандартными средствами Linux (утилита *patch*).

Для применения yaml-файлов используется инструмент *clang-apply-replacements*, также реализованный на базе clang.

3. Подходы к исправлению дефектов

Вообще говоря, не обязательно вносить изменения в код именно в том месте, где статический анализатор обнаружил ошибку. Внесение изменений в других местах порой целесообразнее с точки зрения программиста, но может требовать более глубокой интеграции анализатора и исправителя. В зависимости от сложности определения позиции для модификации подходы к автоматическому исправлению можно классифицировать следующим образом.

- **Исправление в месте обнаружения дефекта:** на вход инструменту автоматического исправления поступает позиция, в которой дефект был обнаружен, тот находит требуемое место в коде (или AST'е) и на основе доступного ему анализа определяет, что и как требуется изменить. Поддержка автоматического исправления для таких дефектов требует минимальных изменений в системе их поиска.
  Пример 4. Дефект "присвоенное переменной значение нигде не используется".
  Очевидное исправление такого дефекта – удалить такое присваивание полностью или частично в зависимости от наличия побочных эффектов. Изменения в каких-то других местах кода не требуются.
  Большинство реализованных в нашем инструменте автоматических исправлений относится к этому типу. Кроме упомянутого в *Примере 4*, это, например, "неинициализированный в конструкторе член класса", "использование небезопасной функции" и "сравнение с всегда истинным/всегда ложным результатом".
- **Исправление, позицию которого можно однозначно определить по трассе дефекта.**
  Пример 5. Дефект "используется неинициализированная переменная".
  В данном случае позиция ошибки – это не то место, которое нужно исправлять. Исправлять нужно декларацию данной переменной. Позиция этой декларации есть в трассе дефекта, и детектор мог бы передать её инструменту исправления. В этом случае нужны изменения в детекторе, но они минимальны и реализовать их несложно.
- **Случаи, когда исправление требуется внести в нескольких не связанных между собой местах кода, но каждое такое место однозначно определяется по трассе дефекта.**
  Пример 6. Рассмотрим следующий случай дефекта "указатель был разыменован, а затем проверен на равенство NULL":

```
 1: char *buf = getbuf();
 2: memset(buf, '\0', 5);
      ...
13: if (! buf) {
14:  goto err;
15: }
```

  Если код в строках 3-12 удовлетворяет определенным условиям, в частности, не содержит присваиваний переменной buf, то адекватным исправлением данной проблемы будет переместить проверку buf на равенство NULL перед вызовом memset. Для этого нужно знать две позиции - разыменования и проверки.
- **Более сложные алгоритмы вычисления позиции исправления. Примерами** могут служить системы MemFix ([1]), выявляющая и исправляющая утечки ресурсов, и VFix ([2]), предназначенная для поиска и исправления разыменований нулевого указателя. В обеих системах в детекторе дефектов происходит не только обнаружение проблемы в коде, но и на основе достаточно сложного потокового анализа определяется оптимальное место её исправления. В нашем случае использование подобных алгоритмов означало бы реализацию новых детекторов дефектов или принципиальную переделку имеющихся,

что выходит за рамки поставленной задачи.

4. Шаблоны исправления

Шаблоном исправления мы будем называть совокупность

- синтаксических конструкций, имеющих требуемые позиции в коде и удовлетворяющих определенным ограничениям, и
- правил модификации этих конструкций.

Например, в *Примере 2* такой синтаксической конструкцией будет if-выражение, условие которого имеет заданную позицию и не имеет побочных эффектов, а соответствующая модификация состоит в удалении этого выражения из кода, причем способ удаления зависит от наличия else-ветки. Как видно из *Примера 6*, иногда дефекту соответствует не одна синтаксическая конструкция, а несколько, не являющихся поддеревьями друг друга.

Иногда к одному дефекту применимы несколько шаблонов исправления. В таких случаях теоретически возможны разные подходы:

- выдать пользователю все сгенерированные исправления, оставив за ним выбор наилучшего варианта;
- автоматически выбрать наиболее подходящий шаблон и применить только его;
- сгенерировать все возможные исправления и выбрать наилучшее (например, на основе тестов); в нынешней системе Svace этот вариант едва ли применим;
- некоторая комбинация перечисленных выше вариантов.

В настоящее время в нашей системе предпочтительным является второй вариант.

В нашем инструменте подлежащие исправлению конструкции ищутся при помощи шаблонов ASTMatcher, а далее к найденным узлам AST'а применяются реализованные на C++ дополнительные проверки. Правила модификации кода тоже запрограммированы на C++.

Существуют инструменты, по сути предлагающие некоторую формализацию нашего понятия шаблона исправления. Это, например, системы, основанные на patch-подобных языках, таких как SmPL ([7], [8]) или PATL ([9]). Есть также инструменты, позволяющие формально описать синтаксическую структуру фрагмента кода до исправления и после него на языке, близком к целевому языку программирования, например, Nobrainer ([10]) или Refaster ([11]). В ряде случаев такие инструменты могут использоваться для автоматического исправления. Однако выразительной мощности основанного на SmPL Coccinelle, по-видимому, не хватит для решения нашей задачи: например, там отсутствует возможность проверки позиции искомой конструкции или некоторых её семантических свойств. Практика использования Nobrainer показала, что при поддержке новых шаблонов также регулярно требовалось расширение его возможностей. Кроме того, ни один из этих языков, по-видимому, не позволяет реализовать шаблон, соответствующий *Примеру 6*, т. е. исправление в не связанных между собой местах кода.

5. Автоматическое исправление разыменований нулевого указателя

Дефект "разыменование нулевого указателя" для пользователей очень важен, поскольку может представлять собой серьезную уязвимость ([12]). Поэтому неудивительно, что пользователи проявляют к его автоматическому исправлению повышенный интерес. Как можно его исправить? Очевидное решение – добавить перед разыменованием проверку указателя на NULL – вызывает много вопросов. Какая часть кода не должна выполняться, когда указатель равен NULL? Что делать, если он равен NULL? В частности, если разыменование происходит внутри return-выражения, какое значение нужно вернуть? Иногда дефект данного типа суть проявление проблем в логике программы, и его исправление на самом деле может быть гораздо сложнее. Рассмотрим

Пример 7.

```
 1: char *default_name = "";
```

```
 2: char *default_string_value = NULL;
 3:
 4: typedef struct {
 5:   char *name;
 6:   char *description;
 7: } described_name;
 8:
 9: void set_default_name(described_name *name) {
10:   // должно быть default_name вместо default_string_value
11:   name->name = default_string_value;
12: }
13:
14: void someproc() {
15:   described_name name;
16:   ...
17:   set_default_name(&name);
18:   ...
19:   if (name.name[0] == '\0') // ALARM!!!
20:     return;
21:   ...
22: }
```

Однако после изучения найденных на реальных проектах дефектов мы пришли к выводу, что существуют синтаксически различимые случаи, когда вполне вероятный вариант исправления предложить можно.

В Svace существует большое количество детекторов, выявляющих те или иные случаи разыменования нулевого указателя, и качество работы некоторых из них довольно высоко. На основании экспертной оценки из этих детекторов было отобрано несколько, выдающих заметный процент автоматически исправимых дефектов, и одновременно предложено несколько шаблонов для исправления этих дефектов. Основой для экспертных оценок, как правило, были результаты анализа на проекте Tizen-5.5.

Отобранные детекторы можно разделить на три типа:

- обнаруживающие потенциальное разыменование нуля путем анализа трассы от присваивания указателю нуля (*source*) до разыменования этого указателя (*sink*);
- обнаруживающие разыменование указателя, когда перед этим в трассе была проверка этого указателя на равенство нулю (DEREF_AFTER_NULL);
- обнаруживающие проверку на равенство нулю указателя, который перед этим был разыменован (NULL_AFTER_DEREF).

Были выделены следующие шаблоны исправления.

1) **FixCondition.** Если разыменование происходит в условии if/for/while, то добавить проверку в начало условия:

   ```
   if (deref(ptr)) ... ⇒
   if (ptr && deref(ptr)) …
   if (!ptr || deref(ptr)) …
   ```

2) **InitWithAlternative.** Разыменование в правой части присваивания или декларации следует заменить на тернарное выражение (если разыменуемый указатель равен 0, присваиваем значение по умолчанию). Как показала практика, данный шаблон следует ограничить только случаями, когда в правой части находится указатель.

   ```
   Data *data = obj->getData(); ⇒
       Data *data = obj ? obj->getData() : 0;
   ```

3) **ReturnWithAlternative.** Аналогичен предыдущему шаблону: разыменование в return-выражении следует завернуть в if-else блок (если указатель равен 0, нужно вернуть значение по умолчанию).

```
return a->data; ⇒
    if (a)
    return a->data;
    else
    return 0;
```

4) **SimpleCheck.** Добавить проверку для инструкции, в которой происходит разыменование:

```
deref(underefable); ⇒
    if (underefable)
    deref(underefable);
```

5) **CheckWithRelated.** Добавить проверку для инструкции, в которой происходит разыменование, и нескольких следующих инструкций, семантически связанных с ней:

```
x = y->f;
z = x + 1;
f(z);
⇒
    if (y) {
    x = y->f;
    z = x + 1;
    f(z);
    }
```

6) **MoveBreakingCheck** (только для дефектов типа **NULL_AFTER_DEREF**). Если простая проверка указателя на равенство 0 содержит выход из программы, функции, цикла, блока, то переместить эту проверку на позицию перед разыменованием:

```
char *buf = (char *)malloc(5);
memset(buf, '\0', 5);
...
if (! buf) {
goto err;
}
⇒
    char *buf = (char*)malloc(5);
    if (! buf) {
    goto err;
    }
    memset(buf, '\0', 5);
    ...
```

7) **MovePositiveCheck** (только для дефектов типа **NULL_AFTER_DEREF**). Если указатель просто проверяется на неравенство 0, то переместить проверку на позицию перед разыменованием:

```
char *buf = (char *)malloc(5);
memset(buf, '\0', 5);
...
if (buf) {
print(buf);
}
⇒
    char *buf = (char *)malloc(5);
    if (buf) {
    memset(buf, '\0', 5);
    ...
    print(buf);
    }
```

Все перечисленные шаблоны (кроме CheckWithRelated) уже реализованы в нашем инструменте.

Очевидно, нередко для одного и того же дефекта будут применимы сразу несколько из этих шаблонов. В системе Svace было принято решение выдавать пользователю не все возможные исправления, а только то, которое соответствует *наиболее приоритетному* шаблону. Когда автоматическое исправление применяется к большим проектам, такое решение снижает нагрузку на пользователя, которому нужно верифицировать сотни вариантов исправления. Шаблоны были упорядочены по приоритету следующим образом:

1) MovePositiveCheck;
2) MoveBreakingCheck;
3) InitWithRelated;
4) FixCondition;
5) ReturnWithRelated;
6) SimpleCheck.

Как было произведено упорядочение? Во-первых, некоторые из рассматриваемых шаблонов не пересекаются по предусловиям (например, MovePositiveCheck/MoveBreakingCheck или FixCondition/ReturnWithRelated), и их относительный порядок не важен. Также более приоритетным считался шаблон с более строгими предусловиями (например, SimpleCheck применим в большинстве случаев, поэтому его приоритет самый низкий). Во всех неоднозначных ситуациях относительный порядок основывался на экспертной оценке.

Результаты автоматического исправления разыменований нуля, найденных в проекте Firefox-53.0.3, представлены в *Табл. 1*.

Табл. 1. Автоматическое исправление разыменований нуля
Table 1. Automatic repair of NULL dereferences

Checker Name	Defects	Fixes	Reviewed	Good	Not Required	Bad	Needs Join	Good Required
NULL_AFTER_DEREF	108	49	49	76%	4%	12%	8%	79%
DEREF_AFTER_NULL	249	109	109	94%	2%	3%	1%	95%
DEREF_AFTER_NULL.EX	471	426	127	11%	81%	5%	3%	58%
DEREF_OF_NULL.EX	78	36	36	44%	39%	14%	3%	73%

Значения столбцов:

- **Defects** – общее число найденных дефектов данного типа;
- **Fixes** – число предложенных исправлений;
- **Reviewed** – число проанализированных исправлений;
- **Good** – доля приемлемых исправлений;
- **Not Required** – доля случаев, когда исправление не нужно (ложное срабатывание детектора);
- **Bad** – доля неверных исправлений;
- **Needs Join** – доля семантически верных исправлений, которые, тем не менее, нужно объединить с соседними (т. е., например, если несколько соседних выражений оборачиваются в одно и то же условие, то следует объединить их в блок, который и поместить под это условие);
- **Good Required** – доля приемлемых исправлений для истинных дефектов.

Таким образом, реализованный алгоритм автоматического исправления дефектов заданных четырех типов показал приемлемое качество для истинных дефектов (см. последний столбец). Однако в ряде случаев процент адекватных исправлений среди всех предложенных вариантов оказался низким из-за некачественной работы самого детектора, т.е. большого числа ложных срабатываний, и отсечь эти ложные срабатывания за счёт предусловий шаблонов исправления также не удалось.

6. Заключение и направление дальнейших исследований

Нами был реализован инструмент, предлагающий варианты исправления для дефектов разных типов, найденных статическим анализатором Svace, как в относительно простых случаях (неинициализированная переменная), так и в довольно сложных (разыменование нулевого указателя). Инструмент показал приемлемое качество работы и производительность.

В дальнейшем предполагается реализовать автоматическое исправление большего числа дефектов. Однако пользователю интересны в первую очередь сложные дефекты (переполнение буфера, утечка ресурсов и т. п.), которые по трудности исправления сопоставимы с разыменованием нуля. Разрабатывать для каждого из них свой набор шаблонов, привлекая экспертов, очень накладно. Поэтому есть надежда генерировать шаблоны для того или иного типа дефектов автоматически. Для этого требуется большой проект, имеющий длинную историю изменений и давно анализируемый анализатором Svace. Идея состоит в том, чтобы сопоставить найденные в коде дефекты с коммитами в репозиторий и получить возможные исправления дефектов того или иного типа. Если полученные исправления можно будет объединить в кластеры, это будут возможные шаблоны автоматического исправления.

Список литературы / References

[1] Lee J., Hong S., Oh H. MemFix: Static analysis-based repair of memory deallocation errors for C. In Proc. of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018, pp. 95-106.

[2] Xu X., Sui Yu. VFix: Value-Flow-Guided Precise Program Repair for Null Pointer Dereferences. In Proc. of the IEEE/ACM 41st International Conference on Software Engineering (ICSE), 2019, pp. 512-523.

[3] Petke J., Haraldsson S.O. et al. Genetic Improvement of Software: A Comprehensive Survey. IEEE Transactions on Evolutionary Computation, vol. 22, no. 3, 2018, pp. 415-432.

[4] Durieux T., Cornu B. Dynamic patch generation for null pointer exceptions using metaprogramming. In Proc. of the IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2017, pp. 349-358.

[5] Belevantsev A., Borodin A. et al. Design and development of svace static analyzers. In Proc. of the Ivannikov Memorial Workshop, 2018, pp. 3-9.

[6] Иванников В.П., Белеванцев А.А. и др. Статический анализатор Svace для поиска дефектов в исходном коде программ. Труды ИСП РАН, том 26, вып. 1, 2014 г., стр. 231-250. DOI: 10.15514/ISPRAS-2014-26(1)-7 / Ivannikov V.P., Belevantsev A.A. et al. Static analyzer Svace for finding defects in a source program code. Programming and Computing Software, vol. 40, issue 5, 2014, pp. 265-275.

[7] Kang H.J., Thung F. et al. Semantic Patches for Java Program Transformation. In Proc. of 33rd European Conference on Object-Oriented Programming (ECOOP), 2019, pp. 1-27.

[8] Padioleau Y., Hansen R.R. et al. Semantic patches for documenting and automating collateral evolutions in Linux device drivers. In Proc. of the 3rd Workshop on Programming Languages and Operating Systems: Linguistic Support for Modern Operating Systems, 2006, 6 p.

[9] Wang C., Jiang J. et al. Transforming Programs between APIs with Many-to-Many Mappings. In Proc. of the 30th European Conference on Object-Oriented Programming (ECOOP), 2016, 26 p.

[10] Савченко В. В., Сорокин К. С. и др. Nobrainer: инструмент преобразования C/C++ кода на основе примеров. Программирование, том 46, no. 5, 2020 г., стр. 33-46 / Savchenko V.V., Sorokin K.S. et al. Nobrainer: A Tool for Example-Based Transformation of C/C++ Code. Programming and Computer Software, vol. 46, no. 5, pp. 362-372.

[11] Wasserman L. Scalable, example-based refactorings with Refaster. In Proc. of the ACM Workshop on Refactoring Tools, 2013, pp. 25-28.

[12] CWE-476: NULL Pointer Dereference. URL: https://cwe.mitre.org/data/definitions/476.html.

Информация об авторах / Information about authors

Сергей Владимирович СЫРОМЯТНИКОВ – научный сотрудник. Сфера научных интересов: статический анализ исходного кода программ, автоматическое исправление дефектов кода.

Sergey Vladimirovich SYROMIATNIKOV – Researcher. Research interests: static analysis of source code, automatic repair of code defects.

DOI: 10.15514/ISPRAS-2021-33(6)-7

Технический долг в жизненном цикле разработки ПО: запахи кода

[1,2] *В.В. Качанов, ORCID: 0000-0002-9371-6483 <vkachanov@ispras.ru>*
[1] *М.К. Ермаков, ORCID: 0000-0002-0483-6097 <mermakov@ispras.ru>*
[1] *Г.А. Панкратенко, ORCID: 0000-0003-3996-8281 <gpankratenko@ispras.ru>*
[1] *А.В. Спиридонов, ORCID: 0000-0001-8374-2453 <aspiridonov@ispras.ru>*
[1] *А.С. Волков, ORCID: 0000-0003-2492-6003 <asvolkov@ispras.ru>*
[1] *С.И. Марков, ORCID: 0000-0002-6687-4937 <markov@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский физико-технический институт, 141701, Россия, Долгопрудный, Институтский пер., д. 9*

Аннотация. Данная статья посвящена обзору наиболее популярных запахов кода, одного из компонентов технического долга, а также методов и инструментов их поиска. В статье проводится сравнительный анализ результатов работы таких инструментов как *DesigniteJava*, *PMD*, *SonarQube*. Инструменты были применены к набору проектов с открытым исходным кодом для вычисления точности обнаружения и согласованности выбранных инструментов. Показаны сильные и слабые стороны подхода, основанного на подсчете метрик кода и отсечения по пороговым значениям, который используется в инструментах. Ручная разметка результатов работы показала низкий процент истинных срабатываний (10% для божественного класса и 20% для сложного метода). Проведён обзор работ, предлагающих усовершенствование стандартного подхода и альтернативные, не использующие метрики. Для оценки потенциала альтернативных подходов разработан прототип обнаружения длинных методов с системой фильтрации ложноположительных срабатываний, использующие методы машинного обучения.

Ключевые слова: запахи кода; технический долг; машинное обучение; метрики кода

Для цитирования: Качанов В.В., Ермаков М.К., Панкратенко Г.А., Спиридонов А.В., Волков А.С., Марков С.И. Технический долг в жизненном цикле разработки ПО: запахи кода. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 95-110. DOI: 10.15514/ISPRAS–2021–33(6)–7

Technical debt in the software development lifecycle: code smells

[1,2] *V.V. Kachanov, ORCID: 0000-0002-9371-6483 <vkachanov@ispras.ru>*
[1] *M.K. Ermakov, ORCID: 0000-0002-0483-6097 <mermakov@ispras.ru>*
[1] *G.A. Pankratenko, ORCID: 0000-0003-3996-8281 <gpankratenko@ispras.ru>*
[1] *A.V. Spiridonov, ORCID: 0000-0001-8374-2453 <aspiridonov@ispras.ru>*
[1] *A.S. Volkov, ORCID: 0000-0003-2492-6003 <asvolkov@ispras.ru>*
[1] *S.I. Markov, ORCID: 0000-0002-6687-4937 <markov@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2] *Moscow Institute of Physics and Technology,*
9, Insitutsky per., Dolgoprudny, 141701, Russia

Abstract. This paper is dedicated to the review of the most popular code smells, which is one of the technical debt components, in addition to methods and instruments for their detection. We conduct a comparative analysis of multiple instruments such as DesigniteJava, PMD, SonarQube. We apply existing tools to set of open-source projects to deduce detection precision and coherence of the selected instruments. We highlight strengths and weaknesses of the approach based on metrics computation and threshold filtering that is used in instruments. Citing of code smells detected by the instruments shows low percentage of true positives (10% for god class and 20% for complex method). We perform literature review of papers suggesting enhancements of the standard approach and alternative approaches that doesn't involve metrics. To assess the potential of alternative methods we introduce our long method detection prototype with a false positive filtering system based on machine learning methods.

Keywords: code smells; technical debt; machine learning; source code metrics.

For citation: Kachanov V.V., Ermakov M.K., Pankratenko G.A., Spiridonov A.V., Volkov A.S., Markov S.I. Technical debt in the software development lifecycle: code smells. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 95-110 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–7

1. Введение

Разработка и поддержка кода программного обеспечения включает в себя значительное количество процессов и активностей, направленных в значительной степени на достижение единственной цели – получить продукт, реализующий требуемую функциональность с определенным уровнем надёжности [1-2]. Сложность ПО, человеческий фактор, а также издержки производства (конкуренция, фиксированные сроки и т.д.) обычно приводят к тому, что в тех или иных аспектах страдает качество разрабатываемого кода [3]. Подобную потерю качества и связанные с ней проблемы описывают термином «технический долг» [1]. Ключевой особенностью наличия технического долга является то, что дальнейшая разработка, связанная с добавлением новой функциональности или исправлением дефектов, требует бóльших усилий, чем это могло было быть. Ошибки проектирования или реализация архитектуры ПО с известными ограничениями (в угоду желанию получить готовый продукт как можно раньше) являются причиной увеличенного расхода времени и трудовых ресурсов на дальнейшую поддержку [3-4].

Разумеется, на практике крайне редко имеется возможность запланировать все возможные требования потенциальных пользователей ПО и предусмотреть все технические проблемы, проявляющиеся при эксплуатации продукта. Более того, реализация сверхгибкой архитектуры, требующая значительного объема ресурсов, является излишней, если подобная гибкость никогда не будет востребована, а самая базовая требуемая функциональность не будет поддержана в разумные сроки. Тем не менее, возможность обнаруживать потенциальные ошибки проектирования непосредственно на этапе разработки является крайне востребованной, особенно если данный процесс будет производиться автоматически,

а решение о том, что делать с обнаруженной проблемой, будет оставаться на усмотрение программиста или системного архитектора [5].

Одним из компонентов технического долга является наличие так называемого «запаха» в программном коде (*code smell*) [2]. Данный термин описывает ряд архитектурных недостатков, обычно затрудняющих понимание, изменение и отладку конкретных фрагментов кода. Стоит отметить, что запахи кода обычно не имеют строгого определения или трактовки, более того, в некоторых случаях их наличие может быть оправдано [6] (например, техническими ограничениями или необходимостью использовать унаследованный код). Тем не менее, существует ряд запахов кода, которые можно считать основополагающими – они являются объектом изучения многочисленных работ по рефакторингу и анализу кода. Для многих из них были предложены правила обнаружения, часть из которых непосредственно используется в инструментах автоматического анализа кода [2].

Фундаментальные и практические исследования в рамках данной области в первую очередь преследуют цель научиться максимально точно и полно определять наличие запахов кода [7-8]. В данной статье будет показано, что несмотря на значительные успехи, доступные на данный момент популярные инструменты обнаружения запахов кода нельзя использовать в качестве универсального оракула, оценивающего является ли определенный фрагмент или компонент кода корректно спроектированным.

Основной причиной такой ситуации является отсутствие фиксированных формальных определений запахов кода, отсюда возникает необходимость использования косвенных признаков в виде определенных атрибутов кода, которые не всегда достаточно полно отражают действительность.

Применение методов машинного обучения для уточнения правил за счёт обработки наборов фрагментов кода, вручную проанализированных экспертами на предмет наличия запахов, является перспективным направлением для преодоления данной проблемы.

В рамках данной статьи проведен обзор наиболее популярных запахов кода, инструментов их обнаружения и исследований, посвящённых задаче повышения точности обнаружения. Разд. 2 содержит некоторое словесное описание общепризнанных запахов кода и стандартные методологии их обнаружения по определенным атрибутам кода. В разд. 3 рассмотрены инструменты обнаружения и представлены результаты анализа набора проектов с открытым исходным кодом; на основе результатов сделаны выводы о соответствии конкретных результатов различных инструментов на одних и тех же целевых проектах и оценка практической пользы от их применения. Разд. 4 посвящён обзору исследований в области анализа кода, нацеленных на уточнение правил обнаружения запахов кода. В разд. 5 представлен прототип инструмента обнаружения длинных методов с системой фильтрации ложноположительных срабатываний, использующей методы машинного обучения. Финальный раздел содержит общие выводы по текущему состоянию области.

2. Запахи кода

Термин запахи кода был предложен Кентом Бэком (Kent Beck) и популяризован Мартином Фаулером (Martin Fowler) [1] в контексте проведения рефакторинга кода – модификации с целью устранения архитектурных недостатков. Каждая разновидность запахов кода описывает определенный набор особенностей, которые в той или иной степени затрудняют работу с кодом.

В качестве иллюстрации того, на каком уровне абстракции задаются запахи кода, рассмотрим следующие примеры [1].

- «Божественный класс» (*God Class*) – компонент, который реализует большое количество разноплановой функциональности. Исправление кода для устранения запаха предполагает разбиение функциональности по принципу «разделяй и властвуй».

- «Завистливый метод» (*Feature Envy*) – метод, который обращается к полям других классов больше, чем к полям собственного класса. Такие ситуации могут свидетельствовать о том, что инкапсуляция данных применяется некорректно, что затрудняет написание тестов и требует создания лишних объектов. Исправление кода в данной ситуации предполагает перенос функции в более подходящий класс.
- «Стрельба дробью» (*Shotgun Surgery*) описывает ситуацию, в которой изменение класса приводит к необходимости делать незначительные изменения в большом количестве других классов. Причинами подобной архитектуры может быть выделение слишком узкоспециализированных сущностей, с которыми производятся одни и те же действия. Для исправления подобной ситуации стоит перенести в один класс все совместно изменяемые методы и поля.
- «Сложный метод» (*Complex Method*) – метод с излишне сложной реализацией (с точки зрения графов потока управления и данных). Понимание и изменение подобных методов потребует больших ресурсов, а реализация тестов может не покрывать все граничные случаи или иметь свою сложную структуру, которая потребует собственного тестирования. Исправление подобного метода предполагает вынесение некоторой части функциональности в отдельный метод.
- «Длинный метод» (*Long Method*) – метод, реализация которого включает слишком большой объем кода (в пересчёте на строки, инструкции и т.д.). Исправление аналогично сложному методу.
- «Длинный список параметров» (*Long Parameter List*) описывает проблему методов и функций, имеющих слишком много формальных параметров. Подобная ситуация может возникать, например, если в метод передают много полей некоторого класса, тогда лучшим решением было бы передать объект класса целиком и получать нужные поля уже внутри метода.

Данный список сформирован на основе следующих принципов – представленные в нём запахи кода фигурируют в большинстве научных работ по данной области и/или могут быть автоматически обнаружены популярными инструментами анализа кода [7-9].

Как видно, описания запахов кода опираются на крайне общие понятия структуры и семантики исходного кода. Обнаружение требует ручного анализа, проводимого экспертом, или использования косвенных характеристик, которые так или иначе отражают «суть проблемы» [6]. В качестве примера рассмотрим божественный класс и попытаемся предположить, по каким характеристикам можно было бы его идентифицировать. По неформальному определению божественный класс реализует большой объем независимой функциональности. С практической точки зрения это как минимум должно означать, что класс определяет большое количество публичных методов. Классы, количество методов которых ниже заданного порога, маловероятно можно будет назвать божественными. Независимость реализуемой функциональности можно определить следующим образом: если класс определяет некоторое количество внутренних полей, а два его метода осуществляют доступ к непересекающимся подмножествам этих полей, то эти методы никак не связаны. Чем меньше степень связности методов класса при большом их количестве, тем более несогласованным является класс. Процесс подсчёта подобных характеристик может быть довольно сложным на практике. Так, например, «геттеры» и «сеттеры» – методы доступа к переменным – вряд ли имеет смысл учитывать при подсчёте связности, потому что обычно они не должны оперировать больше, чем одним полем класса.

В отличие от неформальных определений, подобные характеристики поддаются численному описанию. В контексте области анализа кода подобные числовые характеристики называются термином метрики [10]. Подсчёт метрик может быть легко автоматизирован на основе инструмента, который осуществляет разбор исходного кода. Так, количество методов

класса (NMD – Number of Methods Declared) и степень несогласованности методов (LCOM – Lack of Cohesion of Methods) – метрики, которые часто входят в набор поддерживаемых инструментами анализа кода.

Подход на основе метрик на данный момент является основным среди инструментов, предоставляющих автоматический поиск запахов кода. Конкретный набор метрик, выбор граничных значений и формулы, составленные на основе данных метрик, могут варьироваться и настраиваться на конкретные языки программирования и/или проекты – это и есть основополагающие проблемы данного подхода.

В следующем разделе приведены примеры таких инструментов и анализ результатов их работы.

3. Поиск запахов кода

Определение запахов кода на основе метрик позволяет автоматизировать анализ кода проекта. Данный анализ может быть встроен в процесс непрерывной разработки с целью поддержания качества ПО, однако это оправдано только в том случае, если его применение будет эффективнее ручного труда (с учётом доступных вычислительных и временных ресурсов). Рассмотрим возможную последовательность этапов применения инструмента:

- настройка инструмента для целевой кодовой базы (опционально);
- запуск инструмента и сбор предупреждений о запахах кода;
- ручной анализ каждого предупреждения с целью выбора одной из трёх доступных меток:
  - истинное срабатывание, требует исправления;
  - истинное срабатывание, не требует исправления;
  - ложное срабатывание.
- Дополнительная настройка инструмента с целью уточнения правил (опционально).

Пометка предупреждений «не требует исправления» может быть аргументирована тем, что в данном случае использование «проблемной» архитектуры могло быть сделано осознанно, или же исправление потребует слишком большого количества ресурсов (что противоречит изначальной цели избавиться от технического долга, однако может быть оправдано практическими соображениями). Если инструмент предоставляет некоторые способы настройки на конкретную кодовую базу, то эти настройки должны служить для снижения количества ложных срабатываний. Также неплохой особенностью была бы возможность обучения инструмента игнорировать предупреждения, похожие на те, которые были ранее помечены как «не требующие исправления».

В общем случае, ключевыми метриками практичности инструмента являются количество обнаруживаемых предупреждений и соотношение истинных и ложных срабатываний. Если инструмент не находит никаких предупреждений или находит слишком много ложных предупреждений, то его применение будет негативно сказываться на процессе разработки.

В цели исследования входила оценка следующих аспектов:

- какой объем предупреждений заданных типов выдаёт инструмент?
- каков процент истинных срабатываний среди выданных предупреждений?
- как соотносятся между собой результаты различных инструментов при обработке одной кодовой базы?

3.1 Выбор инструментов для исследования

В рамках данной статьи было проведено исследование набора свободно распространяемых инструментов автоматического поиска запахов кода. Для предварительной оценки были рассмотрены следующие: DesigniteJava, PMD, SonarQube, JDeodorant, SpotBugs, SAD, iPlasma, inFusion, JSpIRIT, DECOR.

DECOR, JSpIRIT, inFusion, iPlasma были отброшены, так как они не были найдены в свободном доступе как полноценные инструменты или дополнения к IDE, либо они больше не поддерживаются. SAD и JDeodorant являются дополнениями к IDE Eclipse, однако в рамках исследования их не удалось настроить для выбранных проектов. Список дефектов, поддерживаемых в SpotBugs, не включает в себя запахи кода, обнаруживаемые другими инструментами. В итоге, для дальнейшего исследования остались 3 инструмента: DesigniteJava, PMD, SonarQube.

PMD [11] – статический анализатор кода, который ищет стандартные ошибки программирования. Этот инструмент поддерживает 8 языков программирования, в том числе Java. В PMD есть некоторый список встроенных проверок кода (около 400 для Java), но также есть возможность их изменения и добавления собственных. Среди встроенных есть детекторы CognitiveComplexity (сложный метод), LongVariable (длинный идентификатор), GodClass (божественный класс), ExcessiveMethodLength (длинный метод).

DesigniteJava [12] – автономный инструмент оценки качества кода, написанного на Java. Версия, распространяемая в свободном доступе, ищет 17 антипаттернов проектирования и 10 запахов реализации кода, в том числе LongMethod (длинный метод), ComplexMethod (сложный метод) и LongIdentifier (длинный идентификатор).

SonarQube [13] – инструмент автоматической проверки кода для обнаружения ошибок, уязвимостей и запахов кода. Инструмент поддерживает 27 языков программирования. Для Java написано 621 правило, среди которых 153 проверки ошибок и 396 проверок запахов кода, в том числе CognitiveComplexity (сложный метод).

Чтобы найти запахи кода, все три инструмента вычисляют метрики и сравнивают эти значения с некоторыми пороговыми. Если порог пересечён, отмечается нарушение.

3.2 Сравнение результатов

Все выбранные инструменты поиска запахов кода рассматривают Java как целевой язык программирования, в связи с чем было принято решение использовать в рамках исследования набор из 9 популярных проектов с открытым исходным кодом на этом языке, а именно: ant[1], drill[2], error-prone[3], ExoPlayer[4], giraph[5], gson[6], guava[7], hive[8], truth[9]. В табл. 1 численные характеристики этих проектов, показывающие объём исходного кода, количество объектов для анализа на наличие антипаттернов. Наличие запахов кода присуще, также, и другим языкам программирования (C++, C#, Python). Однако для этих языков не было найдено достаточное количество инструментов для их рассмотрения.

[1] https://github.com/apache/ant.git, f61ac1296
[2] https://github.com/apache/drill.git, 85d270f3
[3] https://github.com/google/error-prone.git, 54fa3f38
[4] https://github.com/google/ExoPlayer.git, 47b82cdc
[5] https://github.com/apache/giraph.git, 2c63aa23
[6] https://github.com/google/gson.git, f319c1b8
[7] https://github.com/google/guava.git, 3dfd7076
[8] https://github.com/apache/hive.git, de0a7ecb
[9] https://github.com/google/truth.git, eaddc49f

Табл. 1. Характеристики рассматриваемых проектов
Table 1. Projects specifications

Проект	Количество строк кода	Количество пакетов	Количество классов	Количество методов
ant	201k	93	1803	16844
drill	675k	389	6998	79465
error-prone	113k	61	3342	16618
ExoPlayer	238k	88	2038	21592
giraph	112k	134	1750	11125
gson	29k	23	489	3005
guava	548k	31	6840	75049
hive	1412k	590	12661	129015
truth	32k	6	305	4873

Каждый из выбранных инструментов был запущен на всём наборе проектов.

В табл. 2 отображено количество нарушений, которые были найдены инструментами на каждом из проектов. Для проектов ant, ExoPlayer, giraph, guava и hive не удалось настроить SonarQube, поэтому отчеты по этим проектам отсутствуют.

Из всех полученных данных были отфильтрованы только те записи, что относятся к 3 видам запаха кода: длинный метод, божественный класс, сложный метод. Однако даже такой базовый список запахов кода ищут не все инструменты.

Табл. 2. Общее количество срабатываний инструментов на проектах
Table 2. Total number of tool warnings on projects

Проект	**PMD**	**Designite**	**SonarQube**
ant	37261	5954	-
drill	201483	49644	15773
error-prone	54890	11390	2489
ExoPlayer	81847	24769	-
giraph	25436	6698	-
gson	9347	1928	385
guava	223384	52193	-
hive	666423	108106	-
truth	14890	5601	642

Далее на основе отчётов была проведена оценка корреляции результатов работы инструментов. Отсутствие общего стандарта описания ошибок и формирования отчетов усложняет процесс совмещения выявленных нарушений на объектах проекта по файлу/классу/методу.

На рис. 1 и рис. 2 изображены диаграммы Венна, показывающие все пересечения отчётов запахов кода от различных инструментов. Сложные методы ищут все выбранные инструменты, поэтому приведена статистика только для запусков на общих проектах (на которых удалось запустить SonarQube). На рис. 1 видно, что значительная часть записей общая для всех инструментов (374 метода из 1132-х отмеченных хотя бы одним инструментом). DesigniteJava определяет как сложные ещё 422 метода, на которые другие инструменты не отреагировали. Длинные методы SonarQube не определяет, поэтому статистка бралась по всем проектам. Отчет PMD практически включает в себя отчет

DesigniteJava, но также содержит ещё значительное количество записей о других методах. Божественный класс определяет только PMD, поэтому диаграмма Венна для этого запаха кода не приведена.

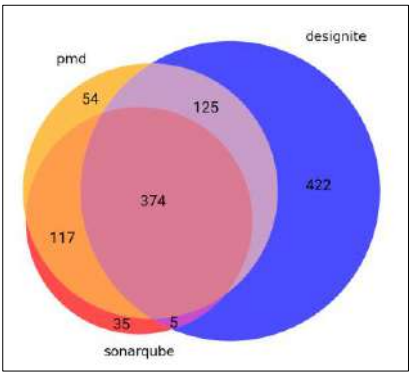


Рис. 1. Диаграмма Венна для «сложного метода»
Fig. 1. Venn diagram for «complex method»

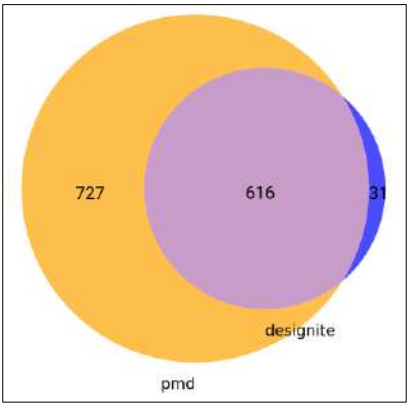


Рис. 2. Диаграмма Венна для «длинного метода»
Fig. 2. Venn diagram for «long method»

3.3 Статистика ложных срабатываний

Для оценки точности обнаружения запахов кода часть предупреждений была исследована вручную. Для каждого типа предупреждений была сформирована случайная выборка из 100 элементов (исключая автоматически сгенерированный и тестовый код), общих для всех инструментов. В рамках ручного анализа каждому предупреждению была назначена одна из трёх оценок – истинное срабатывание, ложное срабатывание и истинное срабатывание, не требующее исправления (англ. won't fix). В табл. 3 представлены результаты анализа. Длинный метод как наиболее «понятный» запах кода, обнаружение которого зачастую связано с количеством строк, имеет наивысший уровень истинных срабатываний, близкий к 100%. С другой стороны, при беглом осмотре части файлов проекта truth было обнаружено много методов, которые эксперты определили как «длинный», но инструменты их не обнаружили. Божественный класс и сложный метод показывают крайне низкий уровень истинных срабатываний (10% и 20% соответственно). Так как процесс оценки дефектов занимает определенное время, подобную статистику можно считать неприемлемой на практике.

Табл. 3. Статистика ложных срабатываний
Table 3. False positive statistics

Оценка эксперта	Божественный класс	Сложный метод	Длинный метод
Истинное срабатывание	10%	20%	90%
Истинное срабатывание, не требует исправления	21%	25%	5%
Ложное срабатывание	69%	55%	5%

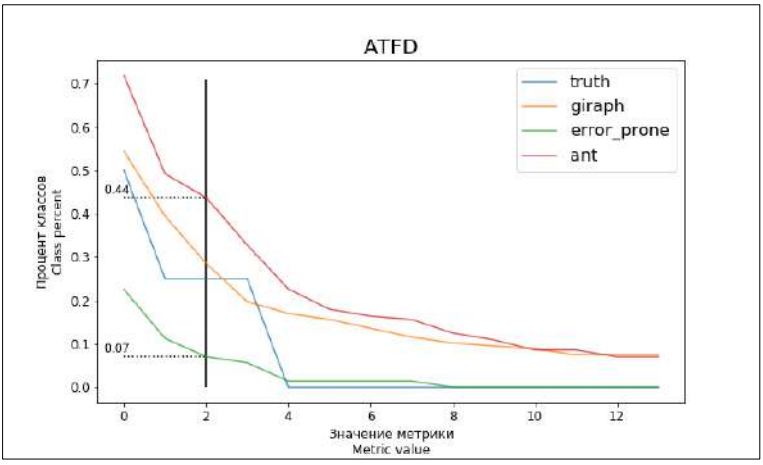


Рис. 3. График распределения классов по метрике ATFD
Fig. 3. The graph of the distribution of classes according to the ATFD metric

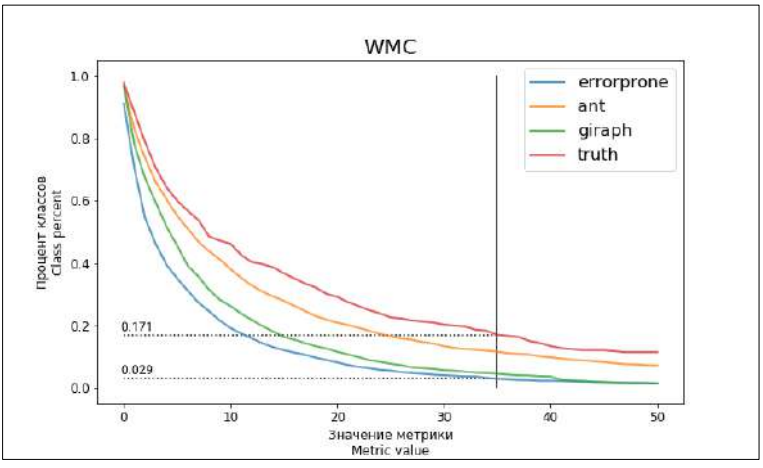


Рис. 4. График распределения классов по метрике WMC
Fig. 4. The graph of the distribution of classes according to the WMC metric

Для демонстрации проблем, связанных с необходимостью выбора пороговых значений, рассмотрим следующую зависимость, которая отображает процент объектов, со значением метрики больше некоторого заданного. На рис. 3, рис. 4 изображены такие графики для метрик ATFD и WMC [14] - базовых метрик для определения божественного класса. ATFD (доступ к сторонним данным) – это количество атрибутов из несвязанных классов, доступ к которым осуществляется напрямую или путем вызова методов доступа. WMC (взвешенные методы класса) – сумма всех сложностей методов в классе. Вертикальной линией отмечены

пороговые значения, взятые из документации PMD[10]. Как видно, при одинаковом пороговом значении процент объектов, превышающих этот порог, а значит определяемых как божественный класс – сильно отличаются, по метрике ATFD 43.75% классов ant, тогда как в error-prone - 7%. Для метрики WMC при установленном пороговом значении error-prone имеет около 3% классов-нарушителей, тогда как truth - чуть больше 17%. Подобное поведение показывает, что пороговые значения необходимо подбирать индивидуально под проект. Рассмотрим также аналогичный график для метрики Cognitive Complexity (некоторая вариация цикломатической сложности, разработанная в SonarQube) рис. 5 – по нему видно, что в проекте error-prone в целом значение этой метрики ниже, чем у drill на 5%, что свидетельствует о различных стилях оформления кода в этой паре проектов.

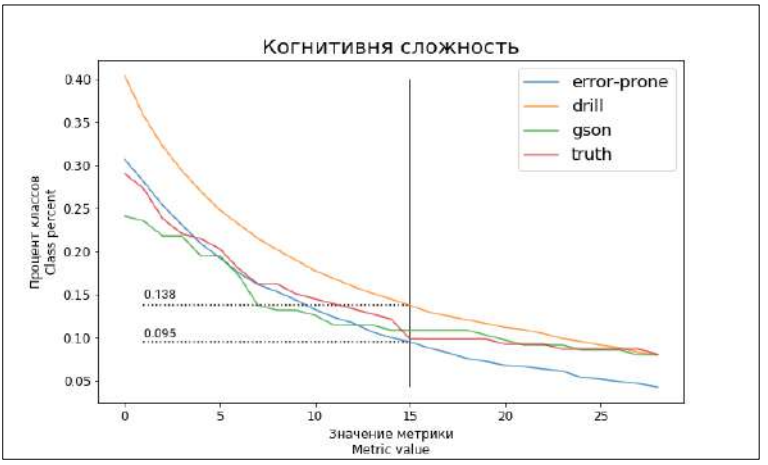


Рис. 5. График распределения методов по метрике когнитивной сложности
Fig. 5. The graph of the distribution of methods according to the cognitive complexity metric

3.4 Ответы на исследовательские вопросы

ИВ1: Какой объем предупреждений заданных типов выдаёт инструмент?

Существующие инструменты при стандартной настройке довольно шумные: PMD выдает предупреждения на 18 - 48% (ant и error-prone) строчек кода проекта (отношение числа репортов к количеству строк кода), Designite чуть сдержаннее – 3 - 17.5% (ant и truth).

ИВ2: Каков процент истинных срабатываний среди выданных предупреждений?

Для божественных классов и сложных методов довольно низкий (10% и 20% соответственно), для длинных методов – 92%, но это не делает данный детектор хорошим, так как полнота его отчёта низкая.

ИВ3: Как соотносятся между собой результаты различных инструментов при обработке одной кодовой базы?

Инструменты имеют значительное пересечение по отчётам на одной кодовой базе, например, 374 из 1132-х общих сложных методов и 616 из 1374 общих длинных методов. Но также имеется и большое количество замечаний, которые пишут одни инструменты, но не другие (рис. 1, рис. 2).

[10] https://pmd.github.io/latest/pmd_rules_java_design.html#godclass

4. Развитие стандартных методов и альтернативные подходы

Как было сказано выше, запахи кода не имеют чёткого определения. Также понятно, что запахи кода имеют сложную структуру и сложные зависимости с другими компонентами кода, что усложняет их идентификацию. Чтобы повысить точность работы рассмотренных инструментов на конкретном проекте необходимо очень точно настроить граничные значения метрик, к тому же, полученные эвристики не могут быть применены к произвольному проекту. Этот процесс может быть очень трудоёмким и неэффективным. Довольно тяжело вручную выбрать более важные метрики и построить правильные эвристики. Использование методов машинного обучения может облегчить задачу и автоматизировать выбор более важных метрик (признаки), а также может находить сложную взаимосвязь между метриками и наличием того или иного антипаттерна в коде.

В [15-16] рассматривается подход использования различных моделей машинного обучения (логистической регрессии, решающих деревьев, метода опорных векторов), основанных на значениях метрик, для определения наличия в данном фрагменте одного из четырёх антипаттернов (божественный класс, класс данных, длинный метод и завистливые функции). В качестве описания одного экземпляра класса используются 63 метрики, для метода – 84. Для обучения моделей был использован набор проектов Qualitas Corpus: 20120401r [17]. Для разметки этих проектов были запущены несколько инструментов (PMD, iPlasma, Fluid Tool) и использованы их отчеты. Оценка качества моделей производилась на наборе примеров, размеченных вручную. Объём размеченного набора: по 420 примеров для каждого типа запаха кода. В результате исследований были определены лучшие модели (J48, Random forest, JRip). Точность определения антипаттернов на предложенном оценочном наборе данных варьируется от 76 до 93 процентов.

В [18] предложено развитие [15-16], а именно применение глубоких нейронных сетей. С помощью автоматического кодировщика [19] вычисленные по коду метрики преобразовываются в новый список атрибутов, далее происходит сокращение размерности полученного вектора, который, в свою очередь, поступает на вход классификаторам, обученным на поиск конкретного запаха кода. Для обучения и тестирования использовался описанный выше [15-16] набор данных. В результате такой обработки метрик авторы статьи смогли определять антипаттерны с точностью от 97 до 99 процентов.

К сожалению, ни тестовый набор данных из [15-16], ни разработанную в [18] модель найти в свободном доступе не удалось.

В [20] предлагают объединить результаты работы нескольких инструментов для более точного поиска антипаттернов. Собранные метрики подаются на вход полносвязной нейронной сети [21]. Наборы данных для обучения и оценки качества были собраны из самих инструментов (DECOR, HIST, JDeodorant, InCode). В результате применения модели к полученным метрикам удалось добиться лучших предсказаний запахов кода. Метрикой качества был выбран коэффициента корреляции Мэтьюса между предсказанными и действительными экземплярами. Для завистливых функций у существующих инструментов лучшие показатели были у InCode – 49%, разработанная модель достигла значения в 56%. Для божественных классов – среди готовых инструментов лучшим был DECOR с показателем в 35%, а авторская модель улучшила этот результат до 49%.

Авторы [22] полагают, что для определения экземпляров запахов кода необходимы не только некоторые абстрактные метрики, но и реальный контекст. Например, методы с конструкцией switch-case будут считаться сложными по версии большинства инструментов, так как цикломатическая сложность превышает некоторый порог. Однако для разработчика, читающего этот код, подобный метод не покажется сложным, потому что все case случаи имеют схожую структуру. В статье проводится исследование о возможности использования токенизированного исходного кода в связке с глубокими нейронными сетями для определения антипаттернов. В качестве первого приближения авторами было предложено

использовать набор данных, размеченный существующим инструментом, работающим на метриках. Хотя, как указывается в статье, лучшим решением было бы использование размеченного вручную набора данных значительных размеров. Для оценки качества полученных моделей был создан небольшой набор примеров (166 классов и 280 методов), размеченных вручную. В результате исследования лучшая из разработанных моделей показала следующие значения F1: 0.64, 0.21 для сложного метода и завистливых функций соответственно.

В ходе обзора существующих решений, основанных на применении машинного обучения, было выдвинуто предположение, что основной проблемой такого подхода является отсутствие большого открытого набора данных, размеченного специалистами, на котором можно проводить общую оценку качества разработанных методов детектирования запахов кода. Эта же мысль озвучена также в [22].

В исследовании [23] авторы постарались решить существующую проблему и создали набор данных из 14739 примеров. Для разметки были выбраны файлы из активных проектов, разрабатываемых на Java, с открытым исходным кодом. Этот набор данных был проанализирован различными специалистами с некоторым процентом перекрестных оценок для повышения качества. Для каждого примера указан тип антипаттерна, его серьёзность и ссылка на сам исходный код. Также указаны некоторые характеристики проверяющих специалистов, такие как уровень образования, длительность работы в области разработки ПО, опыт разработки используя ООП, функциональное программирование, опыт программирования на Java и т.д.

5. Разработанный альтернативный подход

В рамках работы над данной статьёй нами был разработан прототип инструмента обнаружения длинных методов. В данной секции мы рассмотрим его устройство, а также как к нему может быть применена система фильтрации срабатываний, использующая методы машинного обучения.

5.1 Классификатор длинных методов

В качестве алгоритма классификации нами была взята модель произвольного леса деревьев (англ. Random Forest) из библиотеки sklearn[11]. В качестве входа для модели предоставлялись метрики исходного кода, подсчитанные на методах.

Данные для обучения были получены из набора MLCQ, содержащий 2430 уникальных примеров методов. Изначально авторы разделили методы на 4 категории по степени уверенности наличия в примере длинного метода. Далее эта разметка была переведена в бинарный вид наличия или отсутствия запаха кода. А также нами был реализован прототип для сбора метрик исходного кода: LOC (количество строк кода), NCLOC (количество строк кода без комментариев и пустых строк), MCC (цикломатическая сложность Маккаби), TokenCount (количество токенов).

5.2 Фильтрация результатов

После обнаружения моделью длинного метода, при помощи анализа графа потока данных, для каждого блока кода (в случае Java - фрагмент кода, ограниченный фигурными скобками) определяются зависимости переменных внутри этого блока от переменных, находящихся во вне. Далее выбирается блок, в котором зависимых переменных меньше, чем в остальных. Такой блок мы считаем кандидатом для вынесения в отдельную функцию и производим

[11] https://scikit-learn.org

соответствующую трансформацию, а именно создается отдельный файл, класс и метод для этого блока, а в исходном коде на его место вставляется вызов созданного метода.

Далее проводится валидация описанной выше модификации, состоящая из двух шагов.

На первом шаге уже упомянутая модель запускается на обоих участках кода и проверяет, что среди них нет длинных методов. Если данное условие не выполняется, операция выделения откатывается, а предупреждения о том, что исходные метод является длинным, отфильтровывается.

На втором шаге те же участки кода, что и в первом шаге, обрабатываются через модель code2vec[12]. Данная модель по участку кода формирует внутреннее представление, описывающее его семантику, и далее по этому представлению присваивает несколько имён каждому методу. Если для двух участков кода присвоенные их методам имена не пересекаются, мы считаем такие участки семантически различимыми. В случае если выделенный блок текста, помещенный в новый метод нового класса, семантически различим с исходным длинным методом, из которых он был изъят, мы считаем такую трансформацию успешной и, тем самым, подтверждаем, что исходный метод был длинным. В отличном случае, как и в шаге 1, данная трансформация откатывается, а предупреждение о том, что метод является длинным, отфильтровывается.

Детектор длинных методов, реализованный подобным образом, имеет несколько особенностей.

Во-первых, применение методов машинного обучения должно помочь использованию подобного инструмента на различных кодовых базах без дополнительной настройки.

Во-вторых, так как разработчик, оценивая предупреждения рассматривает возможные варианты их исправления, шаги 1 и 2 фильтрации отбрасывают как раз то, что было бы оценено человеком как ложные срабатывания или истинные срабатывания, не требующие исправления.

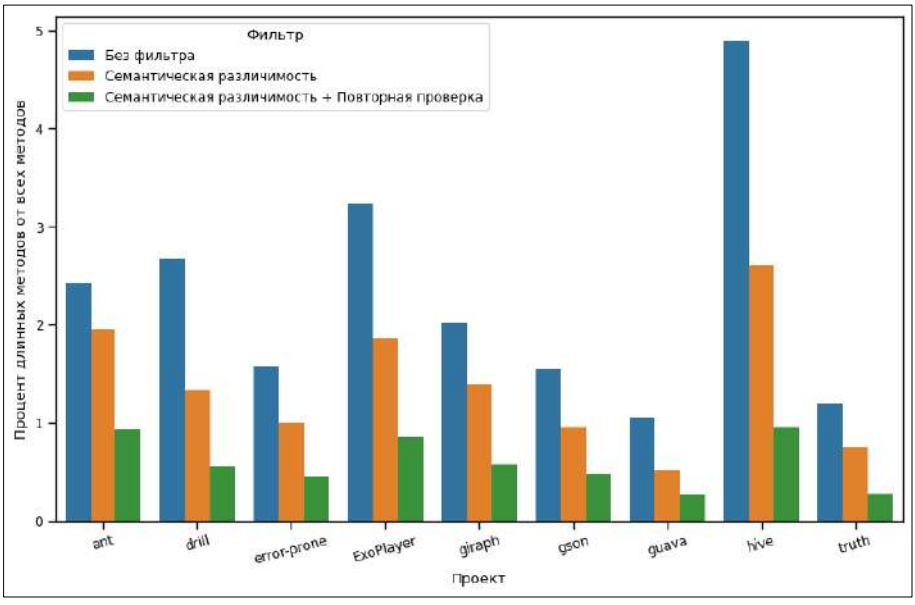


Рис. 6. Эффективность фильтрации длинных методов в зависимости от алгоритма
Fig. 6. Filtering efficiency of long methods depending on the algorithm

[12] https://github.com/tech-srl/code2vec

Для оценки эффективности фильтрации предупреждений о длинных методах мы запустили прототип классификатора на проектах из части 3.2. На результатах классификации была запущена 2-х шаговая фильтрация, описанная выше.

На рис. 6 приведены результаты этой фильтрации. При использовании алгоритма, основанного на семантической различимости выносимого блока от его контекста, прототип отфильтровал 19-50% (проекты ant и guava) предупреждений о длинных методах. Если же помимо указанного алгоритма дополнительно фильтровать те методы, которые после предложенной прототипом трансформации остаются длинными методами, то удастся снизить общее количество предупреждений на 60-80% (проекты ant и hive) от изначального числа.

5.2.1 Оценка качества разработанных методов фильтрации

Из всех полученных базовым классификатором предупреждений были выбраны случайным образом по 30 примеров из каждого проекта. Далее три эксперта разметили эти примеры на три вышеупомянутые категории: истинное срабатывание, ложное срабатывание и срабатывание, не требующее исправления. Для оценки согласованности разметки была использована каппа-статистика Коэна[13]. Коэффициенты парной согласованности: 0.888, 0.826, 0.848. Общий коэффициент согласованности определен как среднее значение парных статистик – 0.85. К разметке были применены описанные выше фильтры: семантическая различимость и семантическая различимость с повторной проверкой классификатором.

В табл. 4 приведены результаты применения фильтров к размеченным данным. В первом столбце указаны типы меток. Во втором и третьем столбцах представлена статистика распределения классов для исходного классификатора. Видно, что к истинным срабатываниям отнесли только 56 (21,2% от всех) предупреждений. В четвертом и пятом столбце показаны данные по работе фильтра, основанного на выделении возможного кандидата и сравнении его семантики с базовым методом. Видно, что он отсеял 60% ложных срабатываний, 38,5% примеров, не требующих исправления, и менее 9% истинных срабатываний. Дополнительная проверка того, остается ли метод длинным, устраняет более 75% ложных срабатываний и тех, что не требуют исправления, но также и более половины (53%) истинных предупреждений. Такое поведение можно объяснить тем, что текущий алгоритм пытается извлечь только один блок операций. В случае очень крупного метода такого исправления может быть недостаточно, чтобы полученный метод больше не считался длинным. Исследование других предикатов фильтрации и извлечения нескольких участков кода выходят за рамки данной работы.

Табл. 4. Статистика качества фильтрации
Table 4. Filtration quality statistics

Метка	До фильтрации, количество	До фильтрации, %	Отсечено фильтром, количество	Отсечено фильтром, % от начального количества
Ложное срабатывание	125	47,4%	75	60%
Не требующее исправления	83	31,4%	32	38,5%
Истинное срабатывание	56	21,2%	5	8,9%
Всего	264		112	

[13] https://en.wikipedia.org/wiki/Cohen%27s_kappa

Таким образом, общее количество длинных методов на рассмотренных проектах оказалось в диапазоне от 0.26% до 0.95% (guava и hive) от общего числа методов на этих проектах. По полученным результатам, мы считаем, что описанный алгоритм фильтрации показал свою эффективность, а итоговое количество длинных методов является достаточно небольшим (в сравнении с количеством всех методов проекта), чтобы охарактеризовать данный подход как «нешумный».

6. Заключение

Запахи кода имеют лишь неформальное определение и могут подвергаться субъективной трактовке авторами различных статей и инструментов. В данной работе проведено сравнение 3 инструментов (DesigniteJava, PMD, SonarQube) обнаружения запахов кода на наборе из 9 проектов (ant, drill, error-prone, ExoPlayer, giraph, gson, guava, hive, truth) с открытым исходным кодом. В силу выбранного авторами подхода с использованием метрик исходного кода и пороговых значений возникает необходимость настройки этих самых пороговых значений под конкретный проект, что показано на рис. 3, 4, 5. Запуск инструментов на реальных проектах показал довольно низкий процент истинных срабатываний (10% и 20% для божественного класса и сложного метода, соответственно). На рис. 1, 2 приведен анализ согласованности отчетов выбранных инструментов.

Также в данной статье проведен обзор существующих предложений по развитию стандартных методов обнаружения с применением методов машинного обучения. По заверениям их авторов, предложенные подходы помогают значительно повысить качество обнаружения запахов кода.

Для проверки гипотезы о потенциале альтернативных подходов в работе разработан прототип инструмента обнаружения длинных методов с системой фильтрации ложноположительных срабатываний, использующие методы машинного обучения. Предложенный алгоритм фильтрации снижает общее количество срабатываний на 60-80%.

Список литературы / References

[1]. Fowler M. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999, 431 p.
[2]. Lanza M., Marinescu R. Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems. Springer, 2010, 231 p.
[3]. van Emden E., Moonen L. Java quality assurance by detecting code smells. In Proc. of the Ninth Working Conference on Reverse Engineering, 2002. pp. 97-106.
[4]. Khomh F., Di Penta M., Gueheneuc Y.G. An exploratory study of the impact of code smells on software change-proneness. In Proc. of the 16th Working Conference on Reverse Engineering, 2009, pp. 75-84.
[5]. Langelier G., Sahraoui H., Poulin P. Visualization-based analysis of quality for large-scale software systems. In Proc. of the 20th IEEE/ACM International Conference on Automated Software Engineering, 2005, pp. 214-223.
[6]. Olbrich S., Cruzes D., Sjberg D. Are all code smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems. In Proc. of the IEEE International Conference on Software Maintenance, 2010, pp. 1-10
[7]. Paiva T., Damasceno A. et al. On the evaluation of code smells and detection tools. Journal of Software Engineering Research and Development, vol. 5, no. 7, 2017, article no. 7.
[8]. Palomba F., Bavota G. et al. Do they really smell bad? a study on developers' perception of bad code smells. In Proc. of the IEEE International Conference on Software Maintenance and Evolution, 2014, pp. 101-110.
[9]. Tufano M., Palomba F. et al. When and why your code starts to smell bad. In Proc. of the IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015, pp. 403-414
[10]. Fenton, N.E., Neil M. Software metrics: successes, failures and new directions. Journal of Systems and Software, vol. 47, no. 2, 1999, pp.149-157.
[11]. PMD. URL: https://pmd.github.io/latest/index.html, accessed: 25/10/2021.
[12]. DesigniteJava. URL: https://github.com/tushartushar/DesigniteJava, accessed: 25/10/2021.
[13]. SonarQube. URL: https://www.sonarqube.org/, accessed: 25/10/2021.

[14]. Ferme V. JCodeOdor: A Software Quality Advisor Through Design Flaws Detection. Master's thesis, Università degli Studi di Milano-Bicocca, Italy, 2013.
[15]. Arcelli Fontana F., Mantyla M., Zanoni M., Marino A. Comparing and experimenting machine learning techniques for code smell detection. Empirical Software Engineering, vol. 21, issue 3, 2016, pp. 1143-1191.
[16]. Arcelli Fontana F., Zanoni M. Code smell severity classification using machine learning techniques. Knowledge-Based Systems, vol. 128, 2017, pp. 43-58.
[17]. Qualitas Corpus. URL: http://qualitascorpus.com/docs/history/20120401.html, accessed: 25/10/2021
[18]. Hamdy A., Tazy M. Deep hybrid features for code smells detection. Journal of Theoretical and Applied Information Technology, vol. 98, 2020, pp. 2684-2696
[19]. Bank D., Koenigstein N., Giryes R. Autoencoders. arXiv:2003.05991, 2021.
[20]. Barbez A., Khomh F., Gu´eh´eneuc Y.G. A machine-learning based ensemble method for anti-patterns detection, arXiv:1903.01899, 2019.
[21]. Zhang P. Neural networks for classification: A survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 30, no. 4, pp. 451-462.
[22]. Sharma T., Efstathiou V., Louridas P., Spinellis D. On the feasibility of transfer-learning code smells using deep learning, arXiv:1904.03031, 2019.
[23]. Madeyski L., Lewowski T. Mlcq: Industry-relevant code smell data set, In Proc. of the International Conference on Evaluation and Assessment in Software Engineering, 2020, pp. 342-347.

Информация об авторах / Information about authors

Владимир Владимирович КАЧАНОВ – аспирант МФТИ, программист в ИСП РАН. Сфера научных интересов: машинное обучение, программная инженерия.

Vladimir Vladimirovich KACHANOV – PhD Student at MIPT, Programmer at ISP RAS. Research interests: machine learning, software engineering.

Михаил Кириллович ЕРМАКОВ – кандидат технических наук, младший научный сотрудник. Сфера научных интересов: динамический анализ кода, фаззинг, символьное исполнение, статический анализ кода.

Mikhail Kirillovich ERMAKOV – Candidate of Technical Sciences, Researcher. Research interests: dynamic program analysis, fuzzing, symbolic execution, static program analysis.

Георгий Александрович ПАНКРАТЕНКО – стажер-исследователь. Сфера научных интересов: программная инженерия, компиляторы.

Georgiy Alexandrovich PANKRATENKO – Intern Researcher. Research interests: program engineering, compilers.

Александр Вячеславович СПИРИДОНОВ – программист. Сфера научных интересов: программная инженерия.

Alexander Vyacheslavovich SPIRIDONOV – Programmer. Research interests: program engineering.

Александр Сергеевич ВОЛКОВ – младший научный сотрудник. Сфера научных интересов: программная инженерия, компиляторы.

Alexander Sergeevich VOLKOV –Researcher. Research interests: program engineering, compilers.

Сергей Игоревич МАРКОВ – научный сотрудник. Сфера научных интересов: статический анализ кода, динамический анализ кода, программная инженерия.

Sergei Igorevich MARKOV – Researcher. Research interests: static program analysis, dynamic program analysis, program engineering.

DOI: 10.15514/ISPRAS-2021-33(6)-8

Сравнение открытых маршрутов проектирования цифровой аппаратуры: qFlow, OpenLANE, Coriolis, SymbiFlow

[1,2,3,4,5] *А.С. Камкин, ORCID: 0000-0001-6374-8575 <kamkin@ispras.ru>*
[1,2] *С.А. Смолов, ORCID: 0000-0003-0173-3081 <smolov@ispras.ru>*
[1,2] *М.М. Чупилко, ORCID: 0000-0002-8772-5631 <chupilko@ispras.ru>*

[1] *Институт системного программирования РАН им. В.П. Иванникова,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2] *Российский экономический университет им. Г.В. Плеханова,*
117997, Россия, г. Москва, Стремянный пер., д. 36
[3] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, г. Москва, Ленинские горы, д. 1
[4] *Московский физико-технический институт,*
114701, Россия, Московская область, г. Долгопрудный, Институтский пер., 9
[5] *НИУ Высшая школа экономики,*
101000, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. В данной работе сделан обзор открытых инструментов логического синтеза, трассировки и размещения элементов моделей цифровой аппаратуры, анализа временных характеристик и синтеза топологических схем. Среди инструментов были выбраны системы проектирования qFlow, OpenLANE, Coriolis и SymbiFlow, поддерживающие полные маршруты: от RTL-модели до двоичных образов для ПЛИС или исходных данных для полупроводниковых фабрик. Для экспериментальной оценки инструментов была взята модель микропроцессора с архитектурой RISC-V под названием PicoRV32. Результаты испытаний показали, что открытые инструменты пригодны для создания топологических схем реалистичных примеров. Однако коммерческие инструменты позволяют создавать более эффективные с точки зрения производительности топологические модели.

Ключевые слова: цифровая аппаратура; микропроцессор; проектирование; открытое программное обеспечение; ПЛИС; СБИС; RISC-V; qFlow; OpenLANE; Coriolis; SymbiFlow

Для цитирования: Камкин А.С., Смолов С.А., Чупилко М.М. Сравнение открытых маршрутов проектирования цифровой аппаратуры: qFlow, OpenLANE, Coriolis, SymbiFlow. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 111-130. DOI: 10.15514/ISPRAS–2021–33(6)–8

Comparison of Open Flows for Digital Hardware Development: qFlow, OpenLANE, Coriolis, and SymbiFlow

[1,2,3,4,5] *A.S. Kamkin, ORCID: 0000-0001-6374-8575 <kamkin@ispras.ru>*
[1,2] *S.A. Smolov, ORCID: 0000-0003-0173-3081 <smolov@ispras.ru>*
[1,2] *M.M. Chupilko, ORCID: 0000-0002-8772-5631 <chupilko@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Plekhanov Russian University of Economics, 36, Stremyanny lane, Moscow, 117997, Russia*
[3] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*
[4] *Moscow Institute of Physics and Technology, 9, Institutskiy per., Dolgoprudny, Moscow region, 114701, Russia*
[5] *National research university Higher school of economics, 20, Myasnitskaya street, Moscow, 101000, Russia*

Abstract. This paper reviews open-source tools for the logical synthesis, place-and-route, static timing analysis and topology generation hardware design stages. The following tools have been described: qFlow, OpenLANE, Coriolis, and SymbiFlow. These tools are aimed to synthesize RTL models into FPGA bitstreams or GDS II physical layouts. A PicoRV32 implementation of RISC-V microprocessor has been used for experimental evaluation of these flows. The results show that open-source flows are capable to produce physical layouts for realistic examples. At the same time, commercial CADs allow generating more effective designs in terms of clock frequency.

Keywords: digital hardware, microprocessor; computer-aided design; open source; FPGA; ASIC; RISC-V; qFlow; OpenLANE; Coriolis; SymbiFlow

For citation: Kamkin A.S., Smolov S.A., Chupilko M.M. Comparison of Open Flows of Digital Hardware Development: qFlow, OpenLANE, Coriolis, and SymbiFlow. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 111-130 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–8

1. Введение

Процесс проектирования аппаратуры включает в себя следующие этапы: (1) логический синтез списка соединений технологических логических элементов из RTL-модели (logic synthesis), (2) размещение логических элементов на схеме и трассировка соединений между ними (place and route), (3) статический временной анализ (static timing analysis), (4) синтез физического топологического представления (physical layout generation).

Входной информацией для первого этапа является RTL-модель аппаратуры, написанная на языке Verilog или VHDL. Логический синтез начинается с синтаксического анализа RTL-модели и составления внутреннего представления, которое затем оптимизируется и преобразуется с помощью технологической библиотеки в список соединений (netlist) логических элементов (gates) или, иначе, ячеек (cells). На втором этапе элементы (ячейки) списка соединений отображаются на ресурсы ПЛИС или размещаются на будущей СБИС. Одновременно происходит трассировка соединений между элементами: сначала общая (global), затем – детальная (detail). На третьем этапе проводится проверка того, что схема после размещения и трассировки удовлетворяет заданным временны́м ограничениям. Если проблем с ограничениями не возникло, осуществляется генерация физического топологического представления аппаратуры.

В случае ПЛИС физическое представление выглядит как двоичный образ, описывающий конфигурации логических элементов ПЛИС (таблиц поиска, lookup tables) и соединения

между ними. В случае СБИС топологическое описание, представленное в виде плоских геометрических фигур в формате GDS II [1], должно пройти этап проверки физических производственных ограничений (design rule checking, DRC) [2]. DRC показывает технологическую корректность топологического описания и его пригодность к использованию на фабрике для производства СБИС.

При получении различных представлений аппаратуры возникает проблема проверки их эквивалентности. Для ее решения существует два подхода.

1) Логическая проверка эквивалентности (Logic equivalence checking, LEC) – используется для проверки отношения соответствия между двумя логическими моделями (например, RTL-моделью и соответствующим списком соединений) [2].
2) Топология-схема (Layout vs schematic, LVS) – используется для проверки того, что физическое топологическое представление (возможно, оптимизированное) эквивалентно исходному списку соединений. Здесь используется обратное построение списка соединений по топологическому представлению [2]. Проверка LVS имеет смысл только при производстве СБИС.

Коммерческие инструменты, с помощью которых в настоящее время доводится до производства основная масса электроники, поддерживают осуществление каждого из перечисленных этапов проектирования аппаратуры. Однако и среди программ с открытым исходным кодом существуют инструменты, решающие как отдельные из рассмотренных задач, так и объединяющие в себе целые маршруты проектирования. Далее в работе такие инструменты будут называться открытыми САПР.

Статья организована следующим образом: во второй главе рассматриваются открытые САПР как для случая СБИС, так и для ПЛИС; в третьей главе описаны открытые инструменты логического синтеза, трассировки и размещения элементов цифровой аппаратуры; в четвертой главе приводятся результаты испытаний САПР на открытой реализации микропроцессора, после чего следуют выводы.

2. Открытые САПР

Основной задачей САПР, рассматриваемых в статье, является преобразование RTL-модели в топологическое представление в формате GDS II или двоичный образ для ПЛИС. В статье рассматриваются следующие системы:

1) для проектирования СБИС: qFlow, OpenLANE, Alliance/Coriolis;
2) для получения двоичных образов для ПЛИС: SymbiFlow.

Основным языком разработки всех рассматриваемых САПР является C++. Сводная информация о рассматриваемых системах представлена в табл. 1.

Табл. 1. Открытые САПР
Table 1. Open-source CADs

Инструмент	**Поддержка**	**Временные рамки разработки**	**Текущая версия**	**Лицензия**
qFlow	Р.Т. Эдвардс	2013 – н. вр.	1.4.98	GPL
OpenLANE	Сообщество разработчиков	2020 – н. вр.	02.00.18	Apache 2.0
Alliance/Coriolis	Сообщество разработчиков	1990 – н. вр.	28.10.2019	GPL, LGPL, Apache 2.0
SymbiFlow	Сообщество разработчиков	2018 – н. вр.	–	ISC

2.1. qFlow

qFlow [3] – открытая САПР, предназначенная для проектирования СБИС по RTL-моделям, написанным на языке Verilog. qFlow включает в себя как сторонние открытые инструменты: Magic, graywolf, qrouter, Yosys, так и небольшие вспомогательные программы, которые запускают сторонние инструменты и конвертируют различные представления аппаратуры. Каждый из отдельных используемых инструментов распространяется под собственной лицензией, а для программного кода, соединяющего эти инструменты в единый маршрут, используется только лицензия GPL [4].

Рассмотрим инструменты, используемые qFlow, упорядочив их по этапам проектирования аппаратуры (иллюстрация схемы работы qFlow версии 1.1, за исключением этапа анализа временных характеристик, находится на рис. 1).

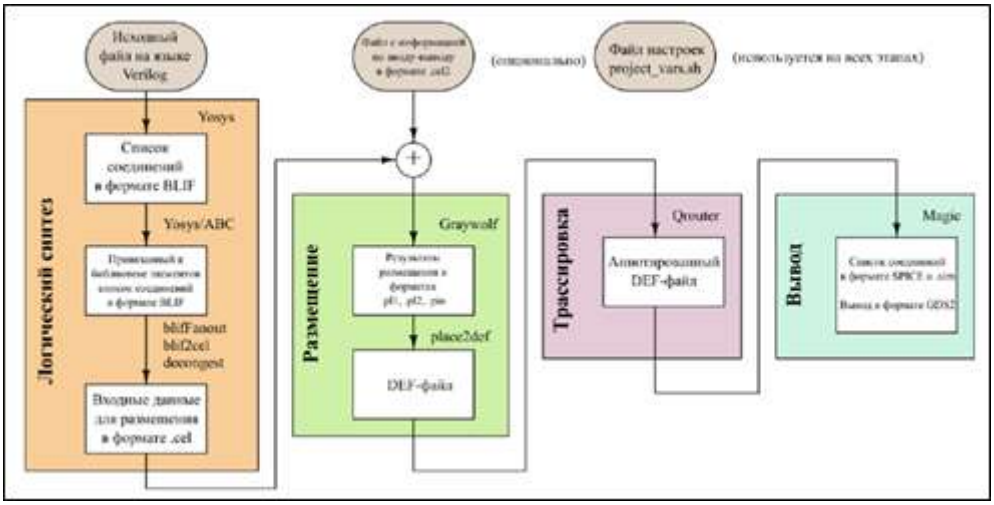


Рис. 1. Схема работы qFlow версии 1.1
Fig. 1. The qFlow 1.1 structure

a. Логический синтез: Yosys [5] или Odin II [6] (оба используют инструмент оптимизации логических схем ABC [7]). Результатом работы является список соединений логических элементов, взятых из технологической библиотеки, в формате BLIF (формат записи логических схем, предложенный Калифорнийским университетом в Беркли (University of California, Berkeley) [8]). С помощью вспомогательного инструмента blif2cel список соединений транслируется в формат описания логических ячеек cel (входной формат для инструмента graywolf).

b. Размещение логических элементов и трассировка соединений: graywolf [9] для размещения и общей трассировки, qrouter [10] для детальной трассировки. Для связи инструментов между собой результат работы graywolf транслируется вспомогательным инструментом place2def в формат DEF (Design Exchange Format) [11] – открытый формат представления модели аппаратуры на топологическом уровне, предложенный компанией Cadence. Инструмент qrouter является простым детальным трассировщиком соединений с ограниченными возможностями, не масштабируемым на сложные промышленные проекты. Результат его работы представляется также в формате DEF.

c. Статический временной анализ: OpenSTA [12] или vesta [13]. Эти инструменты более подробно рассматриваются в разделе 3.4.

d. Физический топологический синтез и DRC: Magic [14], использующий среду моделирования электротехнических характеристик SPICE [15]. Инструмент Magic более

подробно рассматривается в разделе 3.3.

e. LEC/LVS: Yosys для LEC и Netgen [16] для LVS. Эти инструменты более подробно рассматриваются в разделах 3.1 и 3.3 соответственно.

Итак, qFlow поддерживает полный маршрут проектирования от Verilog до GDS II. САПР распространяется с библиотеками логических элементов IIT/OSU 2.7, поддерживающими технологические нормы 180-500 нм, а также с библиотекой FreePDK45 с нормой 45 нм [17], не предназначенной для синтеза GDS II-файлов. qFlow предоставляет пользователю простой графический интерфейс, позволяющий настраивать около десяти параметров синтеза. Отметим, что архитектура qFlow является расширяемой и адаптируемой к новым инструментам и библиотекам элементов. В подтверждение возможностей данной САПР ее автор опубликовал результат синтеза RTL-модели микропроцессора Raven RISC-V [18].

2.2. OpenLANE

OpenLANE [19] – САПР, предназначенная для получения физического топологического представления моделей аппаратуры на основе RTL-моделей, написанных на языке Verilog. OpenLANE использует технологическую библиотеку SkyWater PDK [20], поддерживающую технологическую норму 130 нм. Архитектура OpenLANE близка к qFlow, а одним из основных разработчиков САПР является Р. Т. Эдвардс – создатель qFlow. Как и qFlow, OpenLANE использует различные сторонние инструменты, которые могут распространяться под своими собственными лицензиями.

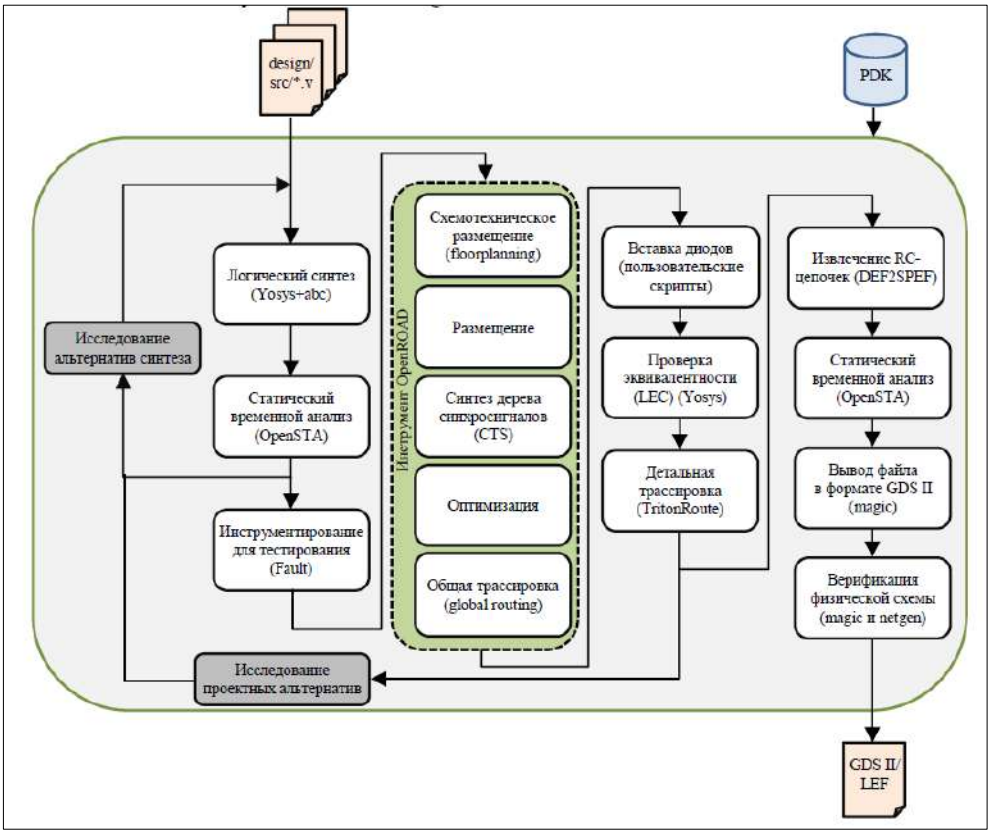


Рис. 2. Схема работы САПР OpenLANE
Fig. 2. The OpenLANE workflow

Рассмотрим основные инструменты, входящие в состав САПР и расположенные по этапам проектирования аппаратуры (см. рис. 2):

a. логический синтез: Yosys и ABC.
b. размещение логических элементов и трассировка соединений: OpenROAD [21], состоящий из нескольких инструментов, выполняющих задачи размещения и общей (глобальной) трассировки, и TritonRoute [22], осуществляющий детальную трассировку.
c. статический временной анализ: OpenSTA.
d. физический топологический синтез и DRC: Magic или KLayout [23];
e. LEC/LVS: Yosys для LEC и Netgen для LVS.

OpenLANE и qFlow используют похожие наборы внешних инструментов. Основными отличиями являются использование в OpenLANE инструментов OpenROAD вместо graywolf и TritonRoute вместо qroute, а также использование библиотеки элементов SkyWater PDK вместо IIT/OSU 2.7 и FreePDK45. Необходимо отметить, что OpenLANE использовалась для производства 40 систем-на-кристалле (СнК) в рамках проекта Open MPW Shuttle Program [24] производства на фабрике SkyWater любительских моделей аппаратуры, спонсируемого компанией Google. Среди выпущенных СнК присутствуют, например, Caravel_RISCV_OSU и Softshell, реализующие системы-на-кристалле с ядрами архитектуры RISC-V.

В целом, САПР OpenLANE выглядит более предпочтительной для промышленного применения, чем qFlow, за счет относительно современных технологических норм библиотеки элементов (130 нм) и более современных специализированных инструментов для этапов размещения и трассировки. Обе указанных САПР используют инструменты Yosys и ABC на этапе логического синтеза.

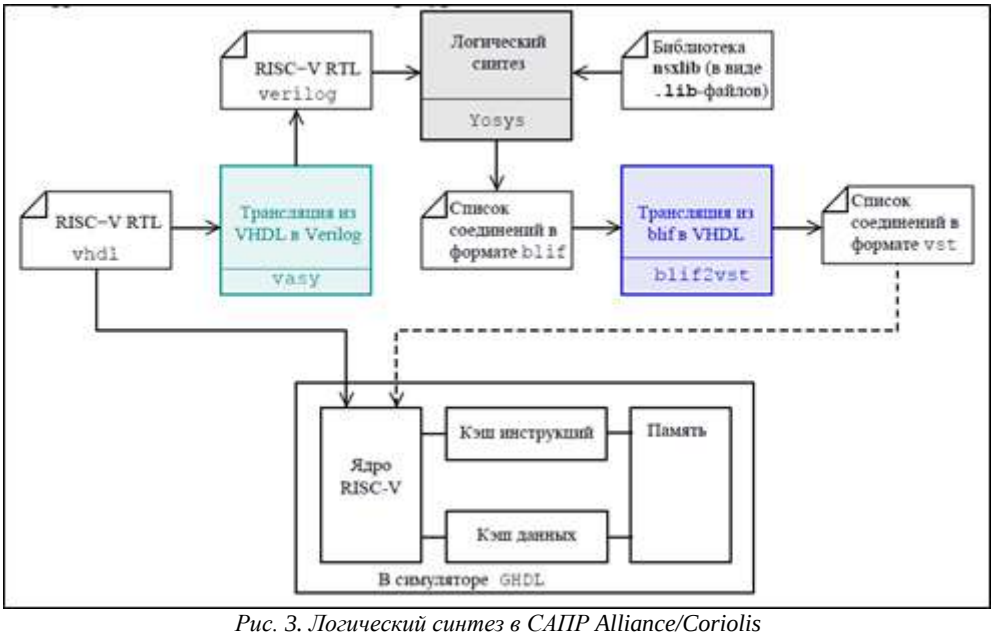


Рис. 3. Логический синтез в САПР Alliance/Coriolis
Fig. 3. Logic Synthesis in Alliance/Coriolis

2.3. Alliance/Coriolis

Alliance/Coriolis [25] – открытая САПР разработки СБИС. Поддерживает полный маршрут получения GDS II-файлов из RTL-моделей, написанных на языке VHDL. Со сменой

собственного логического анализатора Alliance на Yosys САПР стала поддерживать маршрут получения GDS II-файлов из RTL-моделей, написанных на языке Verilog. Отличительной особенностью данной САПР является поддержка разработки как цифровой, так и аналого-цифровой и чисто аналоговой аппаратуры.

Необходимо также заметить, что Alliance/Coriolis поддерживает концепцию OpenAccess [37], предложенную Cadence, а именно позволяет импортировать информацию для размещения и трассировки из LEF- (Library Exchange Format, формат библиотечного обмена) и DEF-файлов (Design Exchange Format, формат обмена моделей) [11].

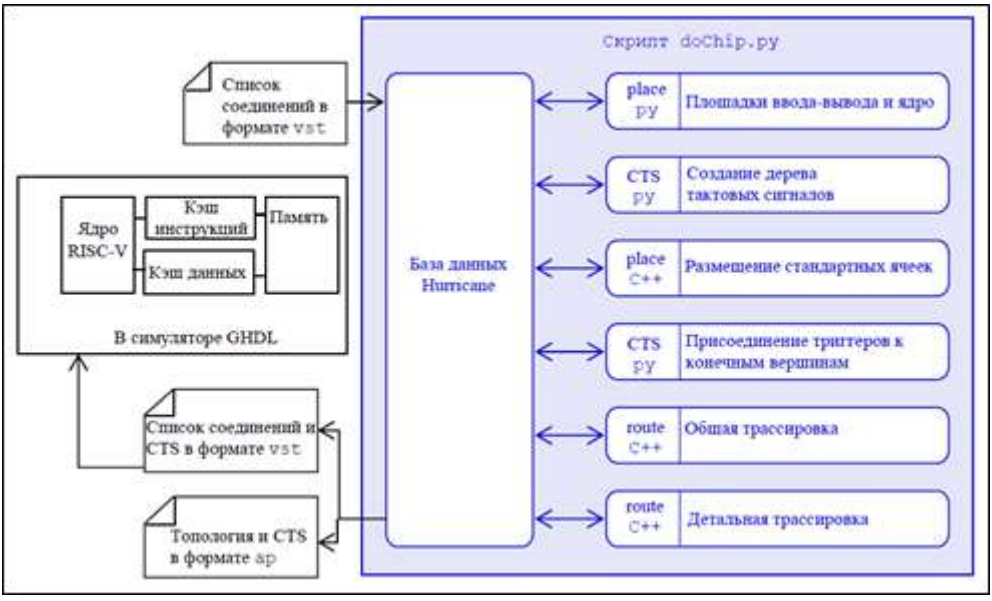


Рис. 4. Размещение и трассировка в САПР Alliance/Coriolis
Fig. 4. Place and route in Alliance/Coriolis

Рис. 5. Этап валидации и получения GDS II-файла в САПР Alliance/Coriolis
Fig. 5. Stages of validation and GDS II generation in Alliance/Coriolis

САПР Alliance/Coriolis применялась в процессе производства ряда сложных моделей аппаратуры, например, для исследовательского суперскалярного микропроцессора StaCS,

включавшего 875 000 транзисторов, и 1-Гигабитного маршрутизатора IEEE Gigabit HSL Router, включавшего 400 000 транзисторов. Однако разработчики САПР отмечают, что Alliance/Coriolis не пригодна для техпроцесса с нормой менее 130-180 нм.

Подводя итог рассмотрения САПР Alliance/Coriolis, подчеркнем, что на протяжении долгого времени (до появления qFlow в начале 2010-х) она являлась единственной открытой системой, реализующей весь маршрут проектирования цифровой аппаратуры. Также отметим, что инструменты (за исключением Yosys), входящие в состав Alliance/Coriolis, являются специфичными для данной среды.

2.4. VTR и SymbiFlow

VTR (Verilog-to-Routing) 8.0.0 [38,39] является открытой САПР, распространяемой по лицензии MIT [40] и предназначенной для подготовки RTL-моделей аппаратуры к размещению на ПЛИС. Ядро VTR – инструмент VPR (Versatile Place and Route) [41] – создается с середины 1990-х годов, при этом сам VTR – с 2010-х годов; последняя версия была выпущена 24.03.2020. Для представления результата работы VTR использует формат FASM (FPGA Assembly) [42], который пригоден для синтеза двоичных образов, загружаемых на ПЛИС, с помощью сторонних инструментов. В качестве такого инструмента создателями VTR предлагается использовать открытую САПР SymbiFlow (см. ниже).

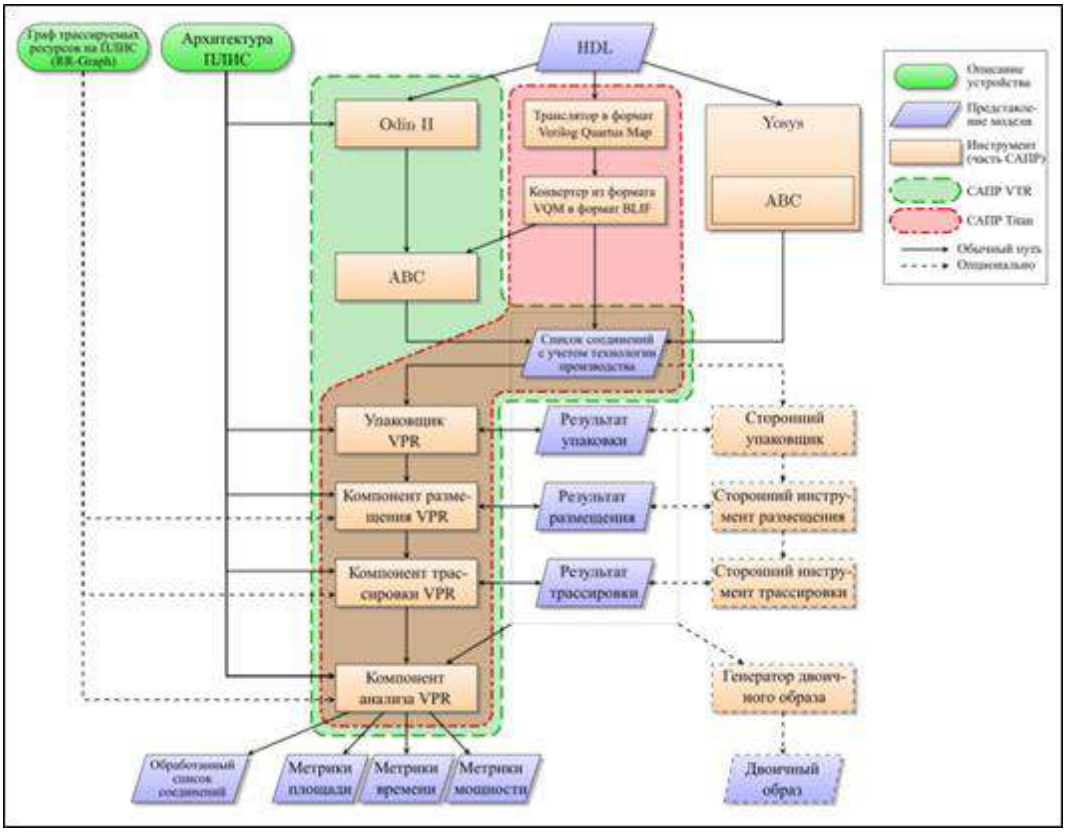


Рис. 6. Схема работы САПР VTR
Fig.6. The VTR workflow

Рассмотрим этапы генерации двоичного образа для ПЛИС с помощью VTR (см. рис. 6, выделение зеленым цветом).

1. Логический синтез: Odin II или Yosys (оба используют ABC). Результат этого этапа представлен в формате BLIF.
2. Размещение и трассировка: VPR. Результат представлен в формате FASM и включает информацию об использовании ресурсов целевой ПЛИС, энергопотреблении, а также временные характеристики.
3. LEC: Yosys.

Некоторые компоненты САПР VTR находят применение и в коммерческих САПР, примером которых может служить Titan [43] (см. рис. 6, выделение красным цветом).

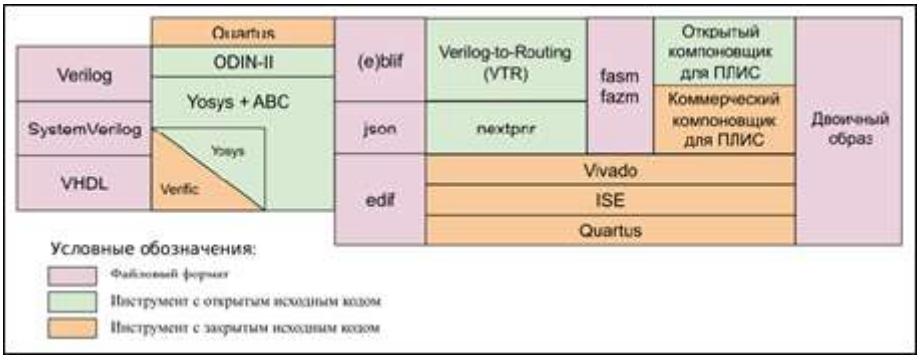


Рис. 7. Схема работы САПР SymbiFlow
Fig. 7. The SymbiFlow workflow

САПР SymbiFlow [44] предназначена для получения двоичных образов, пригодных для загрузки на ПЛИС. Рассмотрим основные этапы работы САПР SymbiFlow для получения двоичных образов (см. рис. 7).

1) Логический синтез: Odin II или Yosys для моделей на языке Verilog; коммерческий инструмент Verific [45] для моделей на языке VHDL.
2) Размещение и трассировка: VTR или nextpnr [46] (оба инструмента создают представление аппаратуры в формате FASM, которое трансформируется в двоичный образ открытым инструментом OpenFPGA [47] или коммерческими инструментами, такими как Vivado [48] и ISE [49] от компании Xilinx, Quartus [50] от компании Intel).

Помимо открытого инструмента OpenFPGA сообщество разработчиков SymbiFlow поддерживает ряд распространяемых по лицензии ISC генераторов двоичных образов с открытым исходным кодом для отдельных серий ПЛИС: Project IceStorm [51] (для ПЛИС Lattice iCE40), Project Trellis [52] (для ПЛИС Lattice ECP5), Project Oxide [53] (для ПЛИС Lattice Nexus), Project Apicula [54] (для ПЛИС Gowin LittleBee), а также Project X-Ray [55] (для ПЛИС Xilinx 7-й серии).

Резюмируя, отметим, что, хотя использование VTR и SymbiFlow позволяет получить двоичный образ, который можно загрузить в некоторые современные ПЛИС, эти САПР едва ли применимы для сложных проектов. Так, в недавней работе [56], посвященной сравнению работы SymbiFlow и коммерческих САПР, был сделан вывод о необходимости доработки алгоритмов размещения и трассировки, используемых в SymbiFlow. В той статье на примере ПЛИС Stratix IV от компании Intel (500 000 LUT-таблиц) было показано, что результаты работы SymbiFlow и САПР Quartus от Intel в целом сравнимы. Однако для ПЛИС Artix-7 от компании Xilinx (50 000 LUT-таблиц), было показано двукратное повышение использование ресурсов ПЛИС по сравнению с САПР Vivado от Xilinx.

3. Инструменты проектирования аппаратуры

Данный раздел включает в себя обзор отдельных инструментов, которые могут быть использованы в открытых маршрутах проектирования. Инструменты сгруппированы по следующим задачам: логический синтез, размещение и трассировка, синтез физического топологического представления и его проверка, статический временной анализ. Отдельно рассматриваются существующие открытые библиотеки элементов.

3.1. Инструменты логического синтеза

В данном подразделе рассматриваются инструменты Yosys, Odin II и GHDL (см. табл. 2).

Yosys 0.10 (Yosys Open SYntethis Suite) от 27.09.2021 – открытый инструмент логического синтеза RTL-моделей, написанных на языке Verilog (поддерживается синтезируемое подмножество языка, описанного в стандарте IEEE 1364-2005), в модели в формате BLIF. Для проведения оптимизаций используется инструмент ABC. Инструмент поддержкой большого количества инструкций Verilog, изначально создан К. Вульфом (C. Wolf).

Odin II (выпущен как часть VTR 8.0.0 от 24.03.2020) – открытый инструмент логического синтеза RTL-моделей, написанных на языке Verilog. Был создан в Университете Нью-Брансуика (University of New Brunswick). Как и в Yosys, в Odin II используется инструмент ABC. Odin II не ориентирован на поддержку RTL-моделей большого размера, поддерживает не все синтезируемые конструкции стандарта IEEE 1364-2005 (например, repeat, deassign, edge, forever, disable), почти не поддерживает конструкции моделирования Verilog на уровне вентилей (за исключением buf) [57].

GHDL 1.0.0 от 03.02.2021 [58] – открытый инструмент логического анализа и моделирования RTL-моделей, написанных на языке VHDL (полностью поддерживается стандарт IEEE 1076-2002, а IEEE 1076-2008 – только частично). Поддержка инструмента осуществляется сообществом разработчиков. Проект был создан Т. Гингольдом (T. Gingold) в середине 2000-х годов. Реализация логического синтеза в данном инструменте находится в экспериментальной стадии и для использования в открытых маршрутах требует доработки.

Заметим, что ни один из открытых инструментов логического синтеза не поддерживает язык SystemVerilog, несмотря на активное его использование в промышленной разработке в целях проектирования и верификации.

Табл. 2. Инструменты логического синтеза
Table 2. Tools for logic synthesis

Инструмент	Входная нотация	Язык разработки	Поддержка	Временные рамки разработки	Лицензия
Yosys	Verilog	C++	Сообщество разработчиков	Начало 2010-х – н. вр.	ISC
Odin II	Verilog	C++	Университет Нью-Брансуика	Середина 2000-х – н. вр.	MIT
GHDL	VHDL	Ada	Сообщество разработчиков	Середина 2000-х – н. вр.	GPL

3.2 Инструменты размещения и трассировки

В данном подразделе рассматриваются следующие инструменты (см. табл. 3): graywolf, OpenROAD, FastRoute, BoxRouter, qrouter, TritonRoute и VPR.

graywolf 0.1.6 от 12.08.2018 – открытый инструмент размещения и общей (глобальной) трассировки, предназначенный для получения логической схемы в формате списка

соединений на основе модели в формате BLIF. Данный инструмент является ответвлением проекта TimberWolf 6.3.5 [59], разработанного в Йельском университете (Yale University).

OpenROAD 2.0 от 17.07.2021 – зонтичный проект для инструментов размещения логических элементов, общей трассировки, создания дерева тактовых сигналов, статического временного анализа и других задач. Поддерживаются форматы LEF, DEF, Verilog. Если считать TritonRoute, описанный ниже, частью данного проекта, то в таком случае OpenROAD поддерживает все преобразования от размещения элементов до детальной трассировки.

Табл. 3. Инструменты размещения и трассировки
Table 3. Tools for place-and-route

Инструмент	Входная нотация	Язык разработки	Поддержка	Временные рамки разработки	Лицензия
graywolf	BLIF	Си	Сообщество разработчиков	1980-е гг. – н. вр.	GPL
OpenROAD	LEF/DEF/Verilog	C++	Сообщество разработчиков	2018 г. – н. вр.	BSD
FastRoute	Текстовый файл	Си	–	2006 г. – 2012 г.	BSD
BoxRouter	Текстовый файл	C++	–	2006 г. – 2007 г.	BSD
qrouter	LEF/DEF	C++	Р.Т. Эдвардс	2011 г. – н. вр.	GPL
TritonRoute	LEF/DEF	C++	Сообщество разработчиков	2018 г. – н. вр.	BSD
VPR	BLIF	C++	Сообщество разработчиков	1997 г. – н. вр.	MIT

FastRoute 4.1 – инструмент общей трассировки с открытой лицензией [60,61]. Использует несколько алгоритмов трассировки, основным из которых является алгоритм волновой трассировки Ли [64]. FastRoute изначально создавался для соревнований между инструментами трассировки, поэтому входные и выходные форматы не являются стандартными. Исходный код инструмента выдается по запросу.

BoxRouter 2.0 – инструмент общей трассировки с открытой лицензией [62,63]. BoxRouter решает задачу трассировки с помощью целочисленного линейного программирования. Инструмент, как и FastRoute, создавался для соревнований, поэтому его входные и выходные форматы не стандартизованы. Исходный код инструмента выдается по запросу.

qrouter 1.4.85 от 22.10.2021 – вспомогательный инструмент для детальной трассировки, разработанный Р. Т. Эдвардсом. Основан на алгоритме волновой трассировки Ли. Работает с файлами в формате LEF и DEF. Результат сохраняется в формате DEF.

TritonRoute 0.0.6.0 от 16.07.2019 – открытый инструмент детальной трассировки, разработанный выпускниками Калифорнийского университета в Сан-Диего (University of California, San Diego). Работает с файлами в формате LEF и DEF. Данный инструмент является основным для детальной трассировки в OpenROAD и, соответственно, в OpenLANE. Основан на представлении задачи трассировки в виде графов, на которых осуществляется минимизация функции стоимости размещения линий [65].

VPR – открытый инструмент размещения и трассировки моделей аппаратуры, разработанный в 1997 г. в Университете Торонто (University of Toronto). Инструмент ориентирован на ПЛИС. Последняя версия была выпущена в рамках САПР VTR 24.03.2020.

3.3. Инструменты синтеза GDS II-файлов и их проверки

В данном подразделе рассматриваются инструменты Magic, KLayout и Netgen (см. табл. 4).

Magic 8.3.220 от 12.10.2021 – открытый инструмент синтеза многослойных топологических схем в форматах GDS II или CIF на основе логических схем, представленных в форматах LEF/DEF. Изначально Magic разрабатывался в Калифорнийском университете в Беркли, а основным текущим разработчиком является Р.Т. Эдвардс. Ограничением инструмента является поддержка только таких проектов, которые содержат горизонтальные и вертикальные линии (так называемый Манхэттенский стиль), однако в некоторых версиях инструмента добавлялась ограниченная поддержка границ другой формы. Инструмент позволяет проводить проверку DRC.

KLayout 0.27.4 от 26.09.2021 – открытый инструмент анализа топологических схем в форматах GDS II или OASIS [66] с возможностью синтеза файлов в формате GDS II. Поддерживает импорт из файлов в формате LEF. Инструмент поддерживает проведение проверок DRC и LVS. Поддерживается только Манхэттенский стиль соединительных линий.

Netgen 1.5.206 от 24.10.2021 – открытый инструмент проведения LVS, в настоящее время используется только для проверки GDS II-файлов, синтезируемых Magic. Изначально разрабатывался М. Сивилотти (M. Sivilotti), текущим разработчиком является Р. Т. Эдвардс.

Табл. 4. Инструменты синтеза GDS II-файлов
Table 4. Tools for GDS II synthesis

Инструмент	Входная нотация	Язык разработки	Поддержка	Временные рамки разработки	Лицензия
Magic	LEF/DEF	Си	Р. Т. Эдвардс	1980-е гг. – н. вр.	Свободная
KLayout	GDS II/LEF	C++	Сообщество разработчиков	2006 г. – н. вр.	GPL
Netgen	GDS II	Си	Р. Т. Эдвардс	2002 г. – н. вр.	GPL

3.4. Инструменты анализа временных характеристик

В данном подразделе рассматриваются инструменты OpenSTA и vesta (см. табл. 5).

OpenSTA 2.2.0 от 14.09.2020 – открытый инструмент, предназначенный для статической проверки временных характеристик на уровне логических вентилей. Основной разработчик – Дж. Черри (J. Cherry). Помимо открытой лицензии может распространяться по коммерческой лицензии от имени компании Parallax Software, Inc., если использующий OpenSTA проект не является проектом с открытым исходным кодом. Входными данными для инструмента являются список соединений, написанный на языке Verilog, временные ограничения в формате SDC (Synopsys Design Constraints) [67], технологическая библиотека в формате Liberty [68]. OpenSTA анализирует задержки различных путей исполнения, при этом находя недостижимые пути.

vesta (qFlow 1.4.98 от 24.09.2021) – открытый инструмент статической проверки временных характеристик, разработанный Р. Т. Эдвардсом как вспомогательный для проекта qFlow. На вход vesta принимает список соединений, написанный на Verilog, и технологическую библиотеку в формате Liberty, а результатом работы является оценка максимальной задержки сигналов в анализируемой схеме и максимальной возможной тактовой частоты.

Табл. 5. Инструменты анализа временных характеристик
Table 5. Tools for static timing analysis

Инструмент	Входная нотация	Язык разработки	Поддержка	Временные рамки разработки	Лицензия
OpenSTA	Verilog	C++	Parallax Software Inc.	2019 г. – н. вр.	GPL

vesta	Verilog	Си	P. T. Эдвардс	2013 г. – н. вр.	GPL

3.5. Библиотека элементов

Библиотеки элементов (Process Design Kit, PDK) включают различную информацию, необходимую для производства СБИС на контрактных фабриках: примитивы логических элементов (символьное представление, параметры), данные для проведения различных проверок (DRC и LVS), технологические данные (допустимые слои, правила их изображения и другое), модели для проведения физического моделирования (SPICE), файлы с дополнительными ограничениями (например, в формате LEF). Полностью укомплектованная PDK должна включать в себя также и GDS II-представления логических элементов.

В данном подразделе рассматриваются следующие библиотеки (см. табл. 6) для технологических норм от 7 до 500 нм: семейство sxlib, IIT/OSU, FreePDK45 (и ее расширение 45nm FreePDK), FreePDK15 (и ее расширение Open-Cell 15nm), SkyWater PDK, ASAP PDK.

Семейство sxlib/ssxlib/vxlib/vsclib/wsclib/vgalib/rgalib/nsxlib [69] – открытые библиотеки элементов, предназначенные для использования в САПР Alliance/Coriolis и созданные в инструменте Graal в рамках работ над указанной САПР. Библиотеки описывают топологию в символическом виде, однако утверждается, что они могут быть отображены на реальный технологический процесс 130 нм, хотя изначально предназначались для 350 нм. Наиболее современной библиотекой, имеющей отношение к данному семейству, является библиотека nsxlib [28], созданная на основе sxlib, проверенная ее создателем на соответствие DRC для FreePDK 45nm, то есть технологическим нормам производства 45 нм.

IIT/OSU 2.7 от 25.06.2007 – набор библиотек, в состав которого входят библиотеки TSMC 180 нм и 250 нм (теоретически подходят для фабрики TSMC, Тайвань), AMI 350 нм и 500 нм. Разрабатывался под руководством проф. Дж. Стина (J. E. Stine, Jr.) сначала в Технологическом институте Иллинойса (Illinois Institute of Technology), а затем в университете штата Оклахома (Oklahoma State University). Библиотеки распространяются в проприетарном формате компании Cadence, однако также включают GDS II-представления логических элементов, а также Verilog- и VHDL-описания элементов для целей симуляции, и используются в qFlow.

FreePDK45 1.4 от 07.04.2011 – библиотека элементов с технологической нормой 45 нм, созданная университетом штата Оклахома и университетом штата Северная Каролина (North Carolina State University, NCSU). Отметим, что FreePDK45, разработанная OSU в 2007-2008 гг., распространялась без лицензии и встроена в САПР qFlow 1.3. Та же самая FreePDK45 параллельно разрабатывалась в NCSU с 2007 г. [70], ее версии 1.0 и 1.1 распространялись по лицензии GPL. FreePDK45 (NCSU) версии 1.2, выпущенная 10.03.2018, уже включала в себя FreePDK45 от OSU. Соответственно, все последующие версии библиотеки выпускались только NCSU и являются частично коммерческими (лицензия Mentor SVRF на использование технологии верификации от компании Mentor), хотя основная часть библиотеки распространяется по лицензии Apache 2.0 (но только в исследовательских целях). Начиная с версии 1.2, появилась расширенная версия библиотеки, называемая 45nm FreePDK [71] и подходящая для организации производства: в ее создании приняли участие коммерческие компании Nangate (в 2019 г. купленная компанией Silvaco) и Semiconductor Research Corporation, некоммерческая организация (альянс) Silicon Integration Initiative (Si2) [72] и фонд поддержки исследований National Science Foundation (NSF) [73]. Библиотека выдается по запросу. Отметим, что в работе [74] говорится о непригодности библиотеки для производства из-за несогласованности значений в библиотечных Liberty и SPICE-файлах.

FreePDK15 1.2 от 09.06.2017 – открытая библиотека элементов, разработанная в NCSU [75]. Является продолжением работ над FreePDK45. Технологический процесс –15 нм.

Распространяется по лицензии BSD. FreePDK15 не включает всю необходимую информацию, необходимую для производственных целей, но существует ее версия, дополненная компанией Silvaco и называемая Open-Cell 15nm [72,76], которая вполне подходит для применения в производстве. Open-Cell лицензируется по Apache 2.0 только для академических целей. Доступ к FreePDK15 и Open-Cell 15nm осуществляется по запросу.

SKY130 PDK 0.0.0 от 08.05.2020 – открытая библиотека элементов, разработанная компанией SkyWater Technology для производства на собственной фабрике при спонсорстве компании Google. Технологический процесс –130 нм. Библиотека поддерживает несколько стратегий размещения элементов на кристалле в зависимости от плотности упаковки, рабочего напряжения и других параметров. Распространяется по лицензии Apache 2.0. Библиотека используется в САПР OpenLANE и проекте MPW производства СБИС, спонсируемом компанией Google.

ASAP 7nm Predictive PDK 1.7 от 02.2021 [77] – открытая библиотека элементов, разработанная в университете штата Аризона (Arizona State University) при участии компании ARM. Технологический процесс – 7 нм, однако библиотека предназначена только для исследовательских целей, но не для производства. Распространяется по некоммерческой лицензии, однако доступна в инструменте OpenROAD (и, соответственно, в САПР OpenLANE по лицензии BSD-3).

Табл. 6. Открытые библиотеки элементов
Table 6. Open technology libraries

Название библиотеки и норма технологического процесса, нм	**Формат описания ячеек (без GDS библиотека не предназначена для производства)**	**Поддержка**	**Временные рамки разработки**	**Лицензия**
Семейство sxlib; 130	Внутренний формат Alliance; формат Synopsis, Verilog	Университет Пьера и Марии Кюри	2002 г. – н. вр.	GPL
IIT/OSU; 180, 250, 350, 500	Форматы Magic и Cadence, GDS, Verilog, VHDL, LEF, Liberty, SPICE	Университет штата Оклахома	1999 г. – 2007 г.	GPL для 1.0, далее – некоммерческая
FreePDK45 / 45nm FreePDK; 45	Форматы Cadence и Synopsis, GDS, Verilog, LEF, Liberty, SPICE	Университет штата Северная Каролина / Silvaco	2007 г. – 2011 г.	GPL до 1.2, далее – Apache 2.0 и SVRF (некоммерческая)
FreePDK15 / Open-Cell 15nm; 15	Форматы Cadence и Synopsis, GDS, Verilog, LEF, Liberty, SPICE	Университет штата Северная Каролина / Silvaco	2014 г. – н. вр.	BSD / Apache 2.0 (некоммерческая)
SkyWater PDK 130nm; 130	Форматы Magic, GDS, Verilog, LEF, Liberty, SPICE	SkyWater Technology	2020 г. – н. вр.	Apache 2.0
ASAP 7nm; 7	Форматы Cadence, Liberty, SPICE	Университет штата Аризона	2016 г. – н. вр.	Некоммерческая (BSD для OpenLANE)

4. Испытания маршрутов проектирования

Для проведения испытаний открытых маршрутов проектирования в рамках данной работы был взят небольшой микропроцессор с открытой архитектурой RISC-V [78] PicoRV32 [79], с объемом кода около 5400 строк, реализующий подмножество инструкций RV32IMC.

В рамках эксперимента мы применили qFlow, OpenLANE, Coriolis для получения топологии в формате GDS II, а также SymbiFlow и коммерческий инструмент Vivado для получения двоичных образов для ПЛИС. Отметим, что САПР VTR не позволила получить FASM-файл для Verilog-модели PicoRV32 из-за ограничений используемого им логического анализатора Odin II. Эксперименты со всеми САПР, кроме Coriolis, проводились на компьютере с центральным процессором Intel Core i7-8700 3.2 ГГц, 32 Гб оперативной памяти и ОС Ubuntu 20.04. Для Coriolis использовалась виртуальная машина с ОС Ubuntu 18.04 и 8 Гб памяти, так как данная САПР не поддерживает работу с ОС Ubuntu 20.04.

Для проведения экспериментов для случая СБИС с получением GDS II-файлов были выбраны следующие библиотеки: OSU350 (350 нм) в qFlow, SKY130 (версия fd_sc_hd, 130 нм) в OpenLANE, символическая cmos45 и реальная nsxlib (130 нм) в Coriolis. Во всех трех САПР использовался Yosys 0.9 для логического синтеза, который, в свою очередь, использует инструмент ABC для привязки к технологическому базису. В табл. 7 приведено сравнение данных для трех открытых САПР и коммерческой САПР [80], построенной на основе среды VSDFLOW [81]. Сравнение осуществлялось по количеству логических элементов схемы, полученному как до привязки к логическим элементам, находившимся в соответствующей библиотеке, так и после нее, по площади на кристалле, по оценке максимальной тактовой частоты. При этом OpenLANE по умолчанию применял опцию flatter (подстановка объектов иерархической структуры для получения «плоской» модели) в Yosys для упрощения RTL-модели до логического синтеза, а qFlow и Coriolis – после синтеза. Для возможности сравнения результатов опции в qFlow и Coriolis были изменены аналогично OpenLANE.

Табл. 7. Результаты экспериментов с PicoRV32 для СБИС
Table 7. Results of Experiments with PicoRV32 targeted at ASICs

Параметры	**САПР**			
	qFlow	**OpenLANE**	**Coriolis**	**VSDFLOW**
Технологическая норма, нм	350	130	130	180
Количество логических элементов до работы ABC	17 040	16 841	16 740	н/д
Количество логических элементов (после отображения на базис с помощью ABC)	25 184	14 840	13 839	13 826
Площадь, мкм^2	12 725 184	442 309	1 803 400	н/д
Оценка максимальной тактовой частоты, МГц	87	24	19	378

Данные, представленные в табл. 7, показывают, что инструменты OpenLANE, Coriolis и VSDFLOW синтезировали схемы с примерно одинаковым количеством логических элементов (после отображения на логический базис с помощью ABC значения для двух последних инструментов отличаются менее чем на 1%, OpenLANE потребовалось на 7% больше элементов, чем Coriolis). Схема, синтезированная инструментом qFlow, включает существенно больше элементов (на 70%), чем в случае OpenLANE. Возможно, это связано с тем, что OpenLANE использует при синтезе большее количество типов логических ячеек, чем qFlow, и, следовательно, потенциально способна проводить более эффективные

оптимизации. Подтверждением этой гипотезы могут служить данные, представленные в табл. 8-9. Так, на примере логической операции AND можно заметить, что инструмент qFlow для ее реализации использует один тип ячеек (AND2X2), а OpenLANE – 5 (and2_2, and3_2, and3b_2, and4_2, and4b_2).

Одновременно, оценочные значения тактовой частоты для OpenLANE и Coriolis существенно ниже, чем для qFlow, что может быть связано с излишним пессимизмом инструментов оценки в первом случае и излишним оптимизмом во втором случае. Оценка частоты проводилась на основе самого длительного времени прохождения данных в модели, подсчитанного инструментами vesta (в qFlow), OpenSTA (в OpenLANE), hitas (в Coriolis). Заметим, что оценочное значение тактовой частоты в коммерческом САПР на порядок выше, чем в открытых инструментах.

Табл. 8. Количество логических ячеек при работе qFlow по типам
Table 8. The number of logical cells in qFlow's result, by types

Тип ячейки				**Количество ячеек**			
AND2X2	808	DFFPOSX1	1 613	NAND3X1	3 009	OAI22X1	263
AOI21X1	2 703	INVX1	2 724	NOR2X1	2 183	OR2X2	330
AOI22X1	444	MUX2X1	303	NOR3X1	281	XNOR2X1	335
BUFX2	339	NAND2X1	3 106	OAI21X1	6 589	XOR2X1	154

Табл. 9. Количество логических ячеек при работе OpenLANE по типам
Table 9. The number of logical cells in OpenLANE's result, by types

Тип ячейки						**Количество ячеек**					
a211o_2	34	a311o_2	2	buf_1	1 653	nor2b_2	1	o221ai_2	11	or2_2	1 091
a211oi_2	59	a31o_2	39	buf_2	8	nor3_2	41	o22a_2	1 314	or2b_2	24
a21bo_2	150	a31oi_2	4	conb_1	42	o2111a_2	1	o22ai_2	52	or3_2	64
a21boi_2	10	a32o_2	39	dfxtp_2	1 613	o211a_2	71	o2bb2a_2	84	or3b_2	5
a21o_2	60	a41o_2	1	inv_2	1 577	o211ai_2	6	o2bb2ai_2	6	or4_2	98
a21oi_2	251	and2_2	158	mux2_1	1 222	o21a_2	53	o311a_2	6	or4b_2	6
a221o_2	87	and3_2	57	mux2_2	2	o21ai_2	139	o31a_2	18	or4bb_2	2
a22o_2	990	and3b_2	1	mux4_1	221	o21ba_2	208	o31ai_2	1		
a2bb2o_2	1 895	and4_2	346	nand2_2	77	o21bai_2	2	o32a_2	117		
a2bb2oi_2	80	and4b_2	2	nor2_2	532	o221a_2	203	o41a_2	4		

Для проведения экспериментов для случая ПЛИС с получением двоичных образов использовались открытая САПР SymbiFlow и коммерческая САПР Vivado 2020.2 от компании Xilinx. В качестве целевой ПЛИС была выбрана модель Xilinx Arty A7-100T.

Табл. 10. Результаты экспериментов с PicoRV32 для ПЛИС
Table 10. Results of Experiments with PicoRV32 targeted at FPGAs

Параметры	**САПР**	
	SymbiFlow	**Vivado**
Использование логических ячеек	4 192 из 63 400 (7%)	2 626 из 63 400 (4%)
Использование блочной памяти, число битов	73 728 из 4 239 360 (2%)	36 864 из 4 239 360 (<1%)
Оценка максимальной тактовой частоты, МГц	62.5	83.3

Результаты экспериментов представлены в табл. 10.

Двоичные образы, синтезированные обеими САПР, используют сопоставимые количества логических ячеек ПЛИС. Максимальная тактовая частота схемы, синтезированной открытой САПР, на 25% ниже, чем у схемы, синтезированной с помощью коммерческой САПР. Что касается использования памяти, то отметим, что на данной ПЛИС она выделяется блоками по 36 Кбит. Это означает, что схема, синтезированная для примера PicoRV32, использовала один (Vivado) или два (SymbiFlow) блока памяти. В целом можно говорить о несколько более высокой эффективности алгоритмов синтеза, реализованных в коммерческой САПР.

5. Выводы

В настоящее время тренд на развитие и использование открытого ПО коснулся также и области проектирования цифровой аппаратуры. В данной статье рассмотрены открытые САПР для проектирования цифровой аппаратуры (qFlow, OpenLANE, Coriolis, VTR, SymbiFlow), а также ряд отдельных инструментов и библиотек, используемых в данных маршрутах. Можно сказать, что открытые САПР способны покрыть весь маршрут проектирования аппаратуры для ПЛИС и его значительную часть – для СБИС. К недостаткам открытых инструментов можно отнести устаревшие технологические нормы используемых библиотек логических элементов, пригодных для создания GDS II-файлов. Также в инструментах отсутствует полноценная поддержка современного языка проектирования и верификации моделей аппаратуры SystemVerilog.

Было проведено экспериментальное сравнение результатов работы данных САПР с коммерческими аналогами на примере небольшого RISC-V ядра PicoRV32. Результаты экспериментов показывают, что среди открытых САПР для СБИС OpenLANE создает топологические представления с наименьшей площадью, а qFlow – с наибольшей максимальной тактовой частотой. По использованию логических элементов (в случае проектирования СБИС) инструменты OpenLANE и Coriolis продемонстрировали результаты, сопоставимые с коммерческой САПР VSDFLOW, а по использованию логических ячеек (в случае проектирования для ПЛИС) открытая САПР SymbiFlow незначительно уступила коммерческому инструменту Vivado. Слабым местом открытых САПР является быстродействие синтезируемых схем – при использовании коммерческих инструментов максимальная тактовая частота, как правило, выше (в 1.3-20 раз).

Список литературы / References

[1]. Graphic Design System User's Operating Manual First Edition. Available at: http://www.bitsavers.org/pdf/calma/GDS_II_Users_Operating_Manual_Nov78.pdf, accessed: 25.10.2021.
[2]. L. Lavagno, G. Martin, L. Scheffer. Electronic Design Automation for Integrated Circuits Handbook – 2 Volume Set. CRC Press, 2006, 1152 p.
[3]. qFlow. Available at: http://opencircuitdesign.com/qflow, accessed: 25.10.2021.
[4]. GNU General Public License. Available at: https://www.gnu.org/licenses/gpl-3.0.ru.html, accessed: 25.10.2021.
[5]. Yosys Open SYnthesis Suite. Available at: http://www.clifford.at/yosys, accessed: 25.10.2021.
[6]. Odin II. Available at: https://docs.verilogtorouting.org/en/latest/odin, accessed: 25.10.2021.
[7]. ABC: A System for Sequential Synthesis and Verification, Available at: http://www.eecs.berkeley.edu/~alanmi/abc, accessed: 25.10.2021.
[8]. Berkeley Logic Interchange Format (BLIF), Available at: http://www.cs.columbia.edu/~cs6861/sis/blif/index.html, accessed: 25.10.2021.
[9]. Graywolf. Available at: https://github.com/rubund/graywolf, accessed: 25.10.2021.
[10]. Qrouter. Available at: http://opencircuitdesign.com/qrouter, accessed: 25.10.2021.
[11]. LEF/DEF Version 5.7. Available at: http://www.ispd.cc/contests/18/lefdefref.pdf, accessed: 25.10.2021.

[12]. OpenSTA. Available at: https://github.com/The-OpenROAD-Project/OpenSTA, accessed: 25.10.2021.
[13]. Vesta. Available at: https://github.com/RTimothyEdwards/qflow/blob/master/src/vesta.c, accessed: 25.10.2021.
[14]. Magic VLSI Layout Tool. Available at: http://opencircuitdesign.com/magic, accessed: 25.10.2021.
[15]. SPICE. Available at: http://bwrcs.eecs.berkeley.edu/Classes/IcBook/SPICE, accessed: 25.10.2021.
[16]. Netgen. Available at: http://opencircuitdesign.com/netgen, accessed: 25.10.2021.
[17]. Oklahoma State University System on Chip (SoC) Design Flows, Available at: https://vlsiarch.ecen.okstate.edu/flow, accessed: 25.10.2021.
[18]. T. Edwards, M. Kassem, C. Wolf. PicoSoC: How we created a RISC-V based ASIC processor using a full open-source foundry targeted RTL-to-GDS flow, and how you can, too! 7th RISC-V Workshop, 2017. Available at: https://riscv.org/wp-content/uploads/2017/12/Wed-1142-RISCV-Tim-Edwards.pdf, accessed: 25.10.2021.
[19]. OpenLANE. Available at: https://openlane.readthedocs.io, accessed: 25.10.2021.
[20]. Open source process design kit for usage with SkyWater Technology Foundry's 130nm node. Available at: https://github.com/google/skywater-pdk, дата обращения 25.10.2021.
[21]. OpenROAD, Available at: https://github.com/The-OpenROAD-Project/OpenROAD, accessed: 25.10.2021.
[22]. UCSD Detailed Router, Available at: https://github.com/The-OpenROAD-Project/TritonRoute, accessed: 25.10.2021.
[23]. KLayout Layout Viewer and Editor. Available at: https://www.klayout.de, accessed: 25.10.2021.
[24]. MPW-ONE Projects. Available at: https://efabless.com/projects/shuttle_name/MPW-ONE, accessed: 25.10.2021.
[25]. Coriolis VLSI CAD Tools. Available at: http://coriolis.lip6.fr, accessed: 25.10.2021.
[26]. M.-M. Louerat, R. Chotin et al. RISC-V design using FOSS. Available at: https://open-src-soc.org/2019-10/media/slides/2nd-RISC-V-Meeting-2019-10-01-10h00-Jean-Paul-Chaput.pdf, accessed: 25.10.2021.
[27]. GNU Lesser General Public License. Available at: https://www.gnu.org/licenses/lgpl-3.0.ru.html, accessed: 25.10.2021.
[28]. N. Shimizu. FOSS for Free HW with Japanese technology process, Available at: https://www.lip6.fr/public/2018-03-09_Shimizu.pdf, accessed: 25.10.2021.
[29]. Hurricane. Available at: https://gitlab.lip6.fr/vlsi-eda/coriolis/-/tree/devel/hurricane, accessed: 25.10.2021.
[30]. Coriolis Etesian, Available at: https://github.com/Coloquinte/Coloquinte, accessed: 25.10.2021.
[31]. Katana. Available at: https://gitlab.lip6.fr/vlsi-eda/coriolis/-/tree/devel/katana, accessed: 25.10.2021.
[32]. FLUTE. Available at: http://home.eng.iastate.edu/~cnchu/flute.html, accessed: 25.10.2021.
[33]. Symbolic Layout. Available at: http://coriolis.lip6.fr/pages/symbolic-layout.html, accessed: 25.10.2021.
[34]. HiTas. Available at: https://www-soc.lip6.fr/en/team-cian/softwares/tasyagle, accessed: 25.10.2021.
[35]. s2r. Available at: https://gitlab.lip6.fr/vlsi-eda/coriolis/-/blob/devel/cumulus/src/plugins/s2r.py, accessed: 25.10.2021.
[36]. druc. Available at: https://gitlab.lip6.fr/vlsi-eda/alliance/-/tree/master/alliance/src/druc, accessed: 25.10.2021.
[37]. OpenAccess. Available at: https://si2.org/oa-tools-utils-libs, accessed: 25.10.2021.
[38]. Verilog-to-Rounting, Available at: https://docs.verilogtorouting.org/en/latest, accessed: 25.10.2021.
[39]. K.E. Murray, O. Petelin et al. VTR 8: High Performance CAD and Customizable FPGA Architecture Modelling. ACM Transaction on Reconfigurable Technology and Systems, vol. 13, issue 2, 2020, Article 9, 55 p.
[40]. The MIT License. Available at: https://opensource.org/licenses/MIT, accessed: 25.10.2021.
[41]. VPR. Available at: https://docs.verilogtorouting.org/en/latest/vpr, accessed: 25.10.2021.
[42]. FPGA Assembly (FASM) Output. Available at: https://docs.verilogtorouting.org/en/latest/utils/fasm, accessed: 25.10.2021.
[43]. K.E. Murray, S. Whitty, S. Liu, J. Luu, V. Betz. Timing-Driven Titan: Enabling Large Benchmarks and Exploring the Gap between Academic and Commercial CAD. ACM Transaction on Reconfigurable Technology and Systems, vol. 8, issue 2, 2015, Article 10, 18 p.
[44]. SymbiFlow. Available at: https://symbiflow.readthedocs.io/en/latest/toolchain-desc/design-flow.html, accessed: 25.10.2021.
[45]. Verific Design Automation. Available at: https://www.verific.com, accessed: 25.10.2021.

[46]. nextpnr. Available at: https://github.com/YosysHQ/nextpnr, accessed: 25.10.2021.
[47]. OpenFPGA. Available at: https://github.com/lnis-uofu/OpenFPGA, accessed: 25.10.2021.
[48]. Vivado ML. Available at: https://www.xilinx.com/products/design-tools/vivado.html, accessed: 25.10.2021.
[49]. ISE Design Suite. Available at: https://www.xilinx.com/products/design-tools/ise-design-suite.html, accessed: 25.10.2021.
[50]. Intel Quartus Prime. Available at: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/download.html, accessed: 25.10.2021.
[51]. Project IceStorm. Available at: http://www.clifford.at/icestorm, accessed: 25.10.2021.
[52]. Project Trellis. Available at: https://github.com/YosysHQ/prjtrellis, accessed: 25.10.2021.
[53]. Project Oxideю Available at: https://github.com/gatecat/prjoxide, accessed: 25.10.2021.
[54]. Project Apicula. Available at: https://github.com/YosysHQ/apicula, accessed: 25.10.2021.
[55]. Project X-Ray, Available at: https://github.com/SymbiFlow/prjxray, accessed: 25.10.2021.
[56]. K.E. Murray et al. SymbiFlow and VPR: An Open-Source Design Flow for Commercial and Novel FPGAs. IEEE Micro, vol. 40, issue 4, 2020, pp. 49-57.
[57]. Odin II Verilog Support. Available at: https://docs.verilogtorouting.org/en/latest/odin/verilog_support, accessed: 25.10.2021.
[58]. GHDL. Available at: https://github.com/ghdl/ghdl, accessed: 25.10.2021.
[59]. C. Sechen, A. Vincentelli. The Timber Wolf Placement and Routing Package. IEEE Journal of Solid-State Circuits, vol. 20, issue 2, 1985, pp. 510-522.
[60]. FastRoute. Available at: http://home.eng.iastate.edu/~cnchu/FastRoute.html, accessed: 25.10.2021.
[61]. M. Pan, Y. Xu et al. FastRoute: an efficient and high-quality global router. VLSI Design, 2012, Article 14, 14 p.
[62]. BoxRouter. Available at: https://www.cerc.utexas.edu/utda/download/BoxRouter.htm, accessed: 25.10.2021.
[63]. M. Cho, K. Lu et al. BoxRouter 2.0: Architecture and Implementation of a Hybrid and Robust Global Router. In Proc. of IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2007, pp. 503-508.
[64]. C.Y. Lee. An Algorithm for Path Connections and Its Applications. IRE Transactions on Electronic Computers, vol. EC-10, issue 3, 1961. pp. 346-365.
[65]. A.B. Kahng, L. Wang, B. Xu. TritonRoute: The Open-Source Detailed Router. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 40, issue 3, 2021, pp. 547-559.
[66]. SEMI P44 – Specification for OASIS. Available at: https://store-us.semi.org/products/p04400-semi-p44-specification-for-open-artwork-system-interchange-standard-oasis-%C2%AE-specific-to-mask-tools, accessed: 25.10.2021.
[67]. Synopsys Design Format. Available at: https://www.intel.com/content/www/us/en/programmable/quartushelp/13.0/mergedProjects/reference/glossary/def_sdc.htm, accessed: 25.10.2021.
[68]. Liberty. Available at: https://people.eecs.berkeley.edu/~alanmi/publications/other/liberty07_03.pdf, accessed: 25.10.2021.
[69]. Alliance standard cell libraries. Available at: http://www.vlsitechnology.org/html/sx_description.html, accessed: 10.11.2021.
[70]. FreePDK45. Available at: https://research.ece.ncsu.edu/eda/freepdk/freepdk45, accessed: 25.10.2021.
[71]. 45nm FreePDK. Open-Cell 15nm, Available at: https://si2.org/open-cell-library, accessed: 25.10.2021.
[72]. Silicon Integration Initiative. Available at: https://si2.org, accessed: 25.10.2021.
[73]. Nangate and Si2 Release Unprecedented Free 45nm Open Source Digital Cell Library. Available at: https://www.src.org/newsroom/press-release/2008/46, accessed: 25.10.2021.
[74]. L.T. Clark, V. Vashishtha et al. Design flows and collateral for the ASAP7 7nm FinFET predictive process design kit. In Proc. of the 2017 IEEE International Conference on Microelectronic Systems Education (MSE), 2017, pp. 1-4.
[75]. FreePDK15. Available at: https://research.ece.ncsu.edu/eda/freepdk/freepdk15, accessed: 25.10.2021.
[76]. M. Martins, J. M. Matos et al. Open Cell Library in 15nm FreePDK Technology. In Proc. of the International Symposium on Physical Design, 2015, pp. 171-178.
[77]. SAP7 PDK. Available at: https://github.com/The-OpenROAD-Project/asap7, accessed: 25.10.2021.
[78]. RISC-V Architecture, Available at: https://riscv.org, accessed: 25.10.2021.

[79]. PicoRV32, Available at: https://github.com/cliffordwolf/picorv32, accessed: 25.10.2021.
[80]. K.P. Ghosh, A.K. Ghosh. Technology mediated tutorial on RISC-V CPU core implementation and sign-off using revolutionary EDA management system (EMS) – VSDFLOW. In Proc. of the China Semiconductor Technology International Conference (CSTIC), 2018, pp. 1-3.
[81]. VSDFLOW. Available at: https://github.com/kunalg123/vsdflow, accessed: 25.10.2021.

Информация об авторах / Information about authors

Александр Сергеевич КАМКИН – кандидат физико-математических наук, ведущий научный сотрудник отдела технологий программирования ИСП РАН, ведущий научный сотрудник научной лаборатории «Гетерогенные компьютерные системы» РЭУ им. Г.В. Плеханова; преподает в МГУ, МФТИ и НИУ ВШЭ. Область научных интересов: автоматизация проектирования цифровой аппаратуры, верификация и тестирование.

Alexander Sergeevich KAMKIN is a leading researcher at the Software Engineering Department ISP RAS, leading researcher at the Plekhanov RUE. He is also a lecturer at MSU, MIPT, and HSE. His research interests include digital hardware design automation, verification and testing. Alexander has a PhD degree in Physics and Mathematics.

Сергей Александрович СМОЛОВ – научный сотрудник отдела технологий программирования ИСП РАН, старший научный сотрудник научной лаборатории «Гетерогенные компьютерные системы» РЭУ им. Г.В. Плеханова. Область научных интересов: автоматизация проектирования цифровой аппаратуры, верификация и тестирование.

Sergey Aleksandrovich SMOLOV is a researcher at the Software Engineering Department of ISP RAS, senior researcher at the Heterogeneous Computing Systems research lab of Plekhanov RUE. His research interests include digital hardware design automation, verification and testing.

Михаил Михайлович ЧУПИЛКО – кандидат физико-математических наук, старший научный сотрудник отдела технологий программирования ИСП РАН, старший научный сотрудник научной лаборатории «Гетерогенные компьютерные системы» РЭУ им. Г.В. Плеханова. Область научных интересов: автоматизация проектирования цифровой аппаратуры, верификация и тестирование.

Mikhail Mikhaylovich CHUPILKO is a senior researcher at the Software Engineering Department of Ivannikov Institute for System Programming of ISP RAS, senior researcher at the Heterogeneous Computing Systems research lab of Plekhanov RUE. His research interests include digital hardware design automation, verification and testing. Mikhail has a PhD degree in Physics and Mathematics.

DOI: 10.15514/ISPRAS-2021-33(6)-9

Обзор методов функционального онлайн-тестирования микропроцессоров

[1]*Н.Д. Черток, ORCID: 0000-0002-7286-7098 <chertoknd@ispras.ru>*
[1,2]*М.М. Чупилко, ORCID: 0000-0002-8772-5631 <chupilko@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Российский экономический университет им. Г.В. Плеханова, 117997, Россия, г. Москва, Стремянный пер., д 36*

Аннотация. Функциональным онлайн-тестированием называется верификация опытных образцов микропроцессоров или их ПЛИС-прототипов, т.е. пост-производственная верификация (post-silicon verification). Такой вид тестирования отличается как от производственного тестирования, нацеленного на проверку работоспособности произведенных микросхем (отсутствие дефектов производства, допустимость значений физических характеристик), так и от функциональной верификации моделей микропроцессоров, проводимой в симуляторе (где можно наблюдать за внутренними сигналами микропроцессора и контролировать процесс исполнения). Пост-производственная верификация позволяет на высокой скорости испытывать огромные массивы тестов и обнаруживать ошибки, пропущенные при функциональной верификации на до-производственном этапе. Тесты для микропроцессоров обычно имеют вид программ; соответственно, основными задачами онлайн-тестирования микропроцессоров являются высокопроизводительная генерация тестовых программ в заданной системе команд и создание тестового окружения, отвечающего за запуск программ, оценку корректности их исполнения микропроцессором, диагностику ошибок и взаимодействие с внешним миром. В данной статье рассматриваются проблемы, возникающие при разработке систем онлайн-тестирования (онлайн-генераторов тестовых программ), делается обзор существующих решений в этой области и на их основе предлагается перспективный подход к организации онлайн-тестирования.

Ключевые слова: микропроцессоры; онлайн-тестирование; функциональное тестирование; пост-производственная верификация; валидация; генерация тестовых программ

Для цитирования: Черток Н.Д., Чупилко М.М. Обзор методов онлайн-генерации тестовых программ для функциональной верификации микропроцессоров. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 131-148. DOI: 10.15514/ISPRAS–2021–33(6)–9

Survey of Methods for Functional Online Testing of Microprocessors

[1]*N.D. Chertok, ORCID: 0000-0002-7286-7098 <chertoknd@ispras.ru>*
[1,2]*M.M. Chupilko, ORCID: 0000-0002-8772-5631 <chupilko@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Plekhanov Russian University of Economics, 36, Stremyanny lane, Moscow, 117997, Russia*

Abstract. Online testing is a process of functional verification of microprocessors produced in silicon or their FPGA-prototypes, i.e. post-silicon verification. This type of testing differs both from the manufacturing testing,

aimed at checking the workability of manufactured chips (e.g., absence of physical defects, admissibility of physical characteristics) and from simulation-based pre-silicon functional verification of microprocessors models (where internal microprocessor signals are available for observing, and the execution process can be controlled). Post-silicon verification enables to rapidly run huge numbers of tests and detect bugs missed during pre-silicon functional verification. Tests for microprocessors are usually represented by executable programs. Accordingly, the main tasks of online testing are high-performance generation of test programs in the given ISA and creation of a test environment responsible for launching programs, assessing the correctness of their execution by a microprocessor, diagnosing errors, and interacting with the outside world. This paper examines the problems arising in the development of online testing systems (online test program generators), reviews existing solutions in this area, and, on the base on them, proposes a promising approach to organizing online testing.

Keywords: microprocessors; online testing; functional testing; post-silicon verification; validation; test program generation

For citation: Chertok N.D., Chupilko M.M. Survey of Methods for Functional Online Testing of Microprocessors. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 131-148 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–9

1. Введение

Долгое время возможности функциональной верификации микропроцессоров, предоставляемые с одной стороны генераторами тестовых программ, а с другой стороны симуляторами RTL-моделей аппаратуры, реализованных на HDL-языках (Register Transfer Level – уровень регистровых передач и Hardware Description Languages – языки описания аппаратуры), были достаточны для проведения функциональной верификации за приемлемое время (до 60% от общего времени разработки [1]). Скорость создания простых тестовых воздействий генераторами тестовых программ можно оценить в десятки-сотни тысяч инструкций в секунду. Если же речь идет о генерации нацеленных (directed) тестов, например, для подсистемы памяти, когда используются сложные ограничения на поток управления и поток данных, скорость создания тестовых воздействий падает до сотен инструкций в секунду. Скорость исполнения тестов на RTL-модели микропроцессора, запущенной в симуляторе, составляет также сотни-тысячи инструкций в секунду. При этом тактовая частота изготовленных микропроцессоров позволяет им работать на несколько порядков быстрее.

Столь существенная разница между скоростью тестирования и реальной производительностью микропроцессоров приводит к необходимости развития средств тестирования, использующих опытные образцы микропроцессоров и их прототипы как для исполнения, так и для генерации тестовых программ. Такого рода средства называются инструментами *онлайн-тестирования* (*online testing*), *пост-производственной валидации* (*post-silicon validation*) [2], или *пост-производственной верификации* (*post-silicon verification*). Процесс проведения онлайн-тестирования обычно осуществляется путем многократного запуска множества разнородных тестов, покрывающих различные функциональные и архитектурные особенности устройства. Поскольку основной целью данного тестирования является поиск функциональных ошибок, допущенных при проектировании, то исполняемые тесты могут быть заимствованы с этапа до-производственного тестирования. Кроме того, тесты можно получить путем автоматизированной генерации на основе *тестовых шаблонов* – параметризованных описаний последовательностей команд, допускающих получение однотипных тестов, обращающихся к различным ресурсам тестируемой системы.

Отметим, что применяющимся на том же этапе изготовления микропроцессоров, но преследующим другие цели, является *производственное тестирование* (*manufacturing*

testing), которое проводится для того, чтобы оценить качество изготовления конкретного микропроцессора и выявить физические ошибки, удовлетворяющие определенному шаблону (замыкание линии на логический 0 или 1, короткое замыкание линий между собой, отказы транзисторов и др.). В производственном тестировании обычной практикой является автоматизированное получение тестов из списка соединений (*netlist*), соответствующего синтезированной RTL-модели тестируемого микропроцессора.

Проведение пост-производственной верификации и разработка соответствующих средств ее проведения сопровождается трудностями в следующих аспектах:

- идентификация и диагностика некорректного поведения;
- оценка полноты тестирования (традиционные техники на основе исходного кода и моделей здесь неприменимы);
- ограничение на сложность инструментов онлайн-тестирования для возможности запуска на ранних этапах пост-производственной верификации;
- необходимость адаптации инструментов онлайн-тестирования под целевую архитектуру;
- применимость к сложным многоядерным и многопоточным микропроцессорам, где необходимо тестировать реализацию механизма поддержки консистентности памяти.

Так, в работе [3] среди проблем, возникающих в процессе онлайн-тестирования, отмечены:

- сложность воспроизведения ошибок (на работу микропроцессора влияет множество условий, включая физический фактор);
- получение эталонных значений (необходима разработка эталонной модели, время работы которой оказывается на порядки больше времени работы тестируемого устройства);
- оценка полноты покрытия (выбор эффективной метрики полноты покрытия является нетривиальной задачей);
- особенности разработки тестовых шаблонов (отсутствие наблюдаемости значений внутренних сигналов тестируемой аппаратуры накладывает дополнительные ограничения на структуру самих тестов и формат результата их выполнения).

Данная работа включает в себя обзор существующих методов проведения онлайн-тестирования и реализующих их средств, а затем на основе обзора предлагает технологию проведения пост-производственного тестирования, которая бы удовлетворяла требованиям архитектурной независимости, скорости работы и позволяла осуществлять проверку корректности наблюдаемого поведения, диагностику ошибок и оценку полноты тестирования.

Далее статья организована следующим образом. Второй раздел посвящен подходам, используемым в различных областях онлайн-тестирования. В третьем разделе рассматриваются наиболее развитые на данный момент инструменты онлайн-тестирования микропроцессоров и проводится сравнительный анализ этих инструментов. В четвертом разделе предлагается архитектурное и технологическое решение для пост-производственной верификации. В заключении приводятся основные тезисы данной работы.

2. Проблемы онлайн-тестирования и методы их решения

В данном разделе приводится обзор основных методологических вопросов проведения пост-производственного тестирования, распределенных по следующим темам:

1) идентификация и диагностика некорректного поведения;
2) оценка полноты тестирования;

3) получение эталонных значений;
4) генерация тестовых данных;
5) применимость к многопоточным микропроцессорам.

2.1. Идентификация и диагностика некорректного поведения

Проблема идентификации и диагностики ошибок является ключевой для пост-производственного тестирования. По сравнению с до-производственной верификацией, осуществляемой с помощью симулятора, где можно в любой момент времени приостановить симуляцию для проверки выполнения заданных свойств, возможности онлайн-тестирования сильно ограничены. Рассмотрим две группы подходов, применяемых для решения данного класса проблем: *использование буферов трасс* (*trace buffers*) и *QED-тесты*.

Начнем рассмотрение проблемы поиска и диагностики ошибок с буферов трасс, как наиболее очевидного решения, изначально использовавшегося только для производственного тестирования. Метод буферов трасс подразумевает добавление в модель аппаратуры специальных накопителей информации, которые сохраняют изменения значений выбранных сигналов для последующего анализа. Использование буферов связано с несколькими направлениями исследований [4]:

- способы выбора сигналов моделей аппаратуры;
- уменьшение количества хранимых данных;
- методы передачи данных в анализирующее программное обеспечение.

Итак, поскольку использование буферов трасс сопряжено с необходимостью хранения и пересылки большого количества информации для ее последующего анализа, очень важным вопросом является уменьшение этого количества за счет выбора адекватного множества отслеживаемых сигналов, а также использования особого способа хранения такой информации. Выбор множества сигналов может быть осуществлен с помощью динамического или статического анализа исходной RTL-модели. Так в работе [5] для выбора сигналов используется симуляция RTL-модели устройства со случайно внедренными ошибками, которая подсказывает какие сигналы наиболее характерны для диагностики некорректного поведения, а в работе [6] предлагается автоматический анализ извлеченного из RTL-кода *графа потока управления и данных* (*Control Data Flow Graph, CDFG*).

В целях уменьшения количества хранимой информации в буферы вместо полных значений сигналов направляется лишь некоторая «*сигнатура*» (*footprint*) [4] – фрагментарная информация об изменениях, как это делается, например, в работе [7] в методе, названном *записью и анализом сигнатур инструкций* (Instruction Footprint Recording and Analysis, IFRA), который мы рассмотрим несколько подробнее. IFRA требует создания дополнительных элементов на синтезируемой схеме (порядка 1% занимаемой площади) для хранения собираемых сигнатур. Структура хранения включает в себя *идентификатор инструкции* (*Instruction ID*) – значение, устанавливающее принадлежность сигнатуры конкретной исполняемой инструкции и зависящее от состояния конвейера целевого микропроцессора: стадии выбора инструкций (fetch) или опустошения конвейера (flush) в результате неверно предсказанного перехода или исключения. Таким образом, для идентификации наблюдаемого поведения используются системные события, связанные с конвейером обработки инструкций. Обнаружение возникающих в процессе работы микропроцессора ошибок осуществляется за счет *пост-триггеров* (*post-triggers*) – дополнительно синтезированных элементов, определяющих возникновение ошибки по микроархитектурному состоянию микропроцессора. При срабатывании пост-триггера текущее состояние сигнатур инструкций сохраняется и используется для воспроизведения ситуации, приведшей к наблюдаемой ошибке, и созданию короткой трассы,

воспроизводящей ошибку. Таким образом, метод IFRA включает в себя подходы как к организации буферов, так и к анализу трасс.

Касаясь темы анализа трасс, рассмотрим еще две работы. Так, проведение анализа ошибочного поведения через воспроизведение ошибок описывается в работе [8], где каждой сохраняемой транзакции в последовательности присваивается метка времени, что позволяет реконструировать осуществляемое взаимодействие в тестируемой аппаратуре, приведшее к ошибке. Кроме того, анализ трасс можно проводить с использованием машинного обучения: в работе [9] используется метод кластеризации для поиска редко случающихся ситуаций в группе однородных трасс, полученных при запуске тестов. Трассы разбиваются на фрагменты, которые группируются в кластеры таким образом, что невозможность отнести новый фрагмент трассы к существующему кластеру сигнализирует о наличии в нем ошибок.

Итак, использование буферов трасс на данный момент является опробованным решением, которое может быть использовано для идентификации и локализации ошибочного поведения, но требует определенного искусства [4] выбора подмножества сигналов, которые будут использоваться при анализе поведения.

Альтернативным решением является метод под названием *локализация ошибок с использованием QED-тестов* (Quick Error Detection, «быстрое обнаружение ошибки»), получаемых в результате применения особого вида трансформаций к существующим тестовым программам с целью уменьшения задержки обнаружения ошибок. Так, можно выделить два родственных подхода: «быстрое обнаружение ошибок электроники» (Electrical Quick Error Detection, E-QED) [10] и «символическое быстрое обнаружение ошибок» (Symbolic Quick Error Detection, S-QED) [11]. Подход E-QED ориентирован на поиск ошибок в микросхемах и включает в себя следующие шаги.

1) Исходный HDL-код автоматически разбивается на блоки, размер которых удовлетворяет ограничениям для запуска в среде инструмента формального анализа. Для каждого блока определяются его интерфейсы. К каждому интерфейсу присоединяется E-QED-блок, записывающий изменения логических значений сигналов в сжатом виде в процессе работы устройства.
2) Запускается набор QED-тестов. При обнаружении ошибки выполнение теста останавливается и считывается содержимое блоков сигнатур.
3) Запускается *BMC-инструмент* (Bounded Model Checking, ограниченная проверка модели) для проверки соответствия сигнатур формальной модели блока. В случае несоответствия блок считается содержащим ошибку.
4) Для каждого содержащего ошибку блока производится серия последовательных запусков BMC-инструмента. При этом в триггеры в составе блока вносятся одиночные однотактовые ошибки («single-cycle FF error model») и проверяется их совпадение с записанной сигнатурой. В случае совпадения данные триггеры считаются содержащими ошибку.
5) Для каждого триггера, содержащего ошибку, проверяется соответствие возникновения ошибки сигнатурам, записанным в соседних блоках. Если соответствие не подтверждается, то триггер исключается из списка содержащих ошибку.

Подход S-QED ориентирован на поиск логических ошибок в работе устройства, а на структуру QED-тестов накладываются дополнительные ограничения, препятствующие генерации тривиальных контр-примеров, не обнаруживающих реальные ошибки. Основной мотивацией создания метода была сложность диагностики ошибок, возникающих при пост-производственной верификации, с помощью традиционных методов, таких как использование буферов трасс. Этот метод включает в себя следующие шаги.

1) В RTL-модель интегрируется модуль, трансформирующий входной поток тестовых инструкций путем применения набора QED-преобразований, обеспечивая выполнение дополнительных ограничений для BMC-инструмента. Интеграция данного модуля необходима только для запуска инструмента и не требуется при синтезе итоговой схемы микропроцессора.
2) Если размер тестируемой схемы превосходит вычислительные возможности BMC-инструмента, то применяется метод моделирования частичного набора компонентов микропроцессора.
3) Система приводится в QED-консистентное состояние – состояние регистров и памяти микропроцессора, при котором исполнение QED-тестов не приведет к генерации ложных контр-примеров BMC-инструментом.
4) Запускается BMC-инструмент для проверки корректности выполнения инструкций путем разрешения ограничений, наложенных на порядок изменения значений регистров, состояние памяти и состояний контрольных модулей. При нарушении какого-либо проверяемого свойства инструмент строит контр-пример, который содержит минимальное количество инструкций, вызывающих его нарушение.

Основанные на QED-тестах подходы видятся перспективными в силу формального характера результатов их работы, но при этом они ограничены возможностями используемого BMC-инструмента, поэтому они являются менее универсальными, чем буферы трасс.

2.2 Оценка полноты тестирования

Полнота тестирования – это метрика, которая позволяет оценить качество проверки RTL-модели микропроцессора, а также качество проверки аппаратной реализации этой модели.

В случае традиционного до-производственного тестирования существует несколько вариантов оценки полноты тестирования, среди которых можно выделить *структурные метрики*, такие как покрытие кода, ветвей, условий и т. д. Однако структурные метрики никак не выражают покрытие функциональных требований спецификации RTL-модели, поэтому для более тщательного тестирования необходимо использовать *поведенческие* (*функциональные*) *метрики*, оценивающие покрытие на основе:

- состояний и переходов конечных автоматов;
- последовательностей переключений уровней сигналов;
- комбинаций входных значений комбинаторных схем.

Структурные метрики не применимы к пост-производственному случаю [12]: при отображении RTL-модели в ее низкоуровневую реализацию изменяется объект тестирования: происходит переход от синтаксических конструкций RTL-модели к соединениям и элементам электронной схемы. А вот функциональные метрики остаются применимыми в силу их более высокой абстракции.

Среди используемых на практике функциональных метрик, наиболее часто встречается проверка *темпоральных* (*временных*) *утверждений*. Для использования этих утверждений в пост-производственном тестировании необходимо иметь способ построения компонентов, отвечающих за непосредственную оценку покрытия. Например, на этапе до-производственного тестирования можно использовать автоматизированное построение компонентов-сборщиков покрытия RTL-модели, их внедрение в RTL-модель, проведение пост-производственного тестирования и анализ полученных результатов на инструментальной ЭВМ. Однако отметим, что синтез и размещение компонента сборки и оценки покрытия вместе с проверяемой схемой может увеличить накладные расходы на занимаемую площадь, потребовать дополнительного питания и изменить временные характеристики схемы.

В работах [13-15] предлагаются различные усовершенствования данного базового метода:

1) в работе [13] для тестирования покрытия требований, накладываемых на протоколы взаимодействия RTL-моделей, предлагается использовать настроенный на тестируемые протоколы анализатор потоков данных, оформленный в виде стороннего IP-блока – необходимость синтезировать компонент-сборщик покрытия отсутствует;
2) в работе [14] предлагается использовать метрику, основанную на покрытии темпоральных утверждений SystemVerilog: при анализе RTL-модели происходит выбор множества задействованных в темпоральных утверждениях сигналов для буферов трасс, а затем информация об изменении значений этих сигналов в процессе онлайн-тестирования используется для реконструкции покрытия утверждений.
3) в работе [15] так же предлагается оценивать полноту тестирования по покрытию темпоральных утверждений, однако утверждения автоматически формируются в процессе синтеза RTL-модели, а факт их покрытия определяется на основании анализа трасс методами работы с большими данными (кластеризация).

Из описания приведенных работ видно, что помимо компонентов, создаваемых специально для анализа покрытия, можно воспользоваться также другими компонентами инфраструктуры пост-производственной верификации: например, буферами трасс и мониторами производительности. Эти идеи озвучены в работе [16] и в книге [17].

Таким образом, можно сделать вывод, что для получения информации о функциональном покрытии RTL-модели в зависимости от сложности этой модели используются:

- средства получения и доставки информации о покрытии (буферы трасс или специально создаваемые компоненты);
- набор темпоральных утверждений, получаемых на каком-либо этапе преобразования RTL-модели;
- инструментальная ЭВМ для анализа покрытия.

2.3 Получение эталонных значений

В связи со значительно более высокой скоростью исполнения тестов в процессе пост-производственного тестирования по сравнению с до-производственной верификацией возможно также значительно увеличить и количество тестовых последовательностей. Последнее теоретически позволяет исследовать поведение тестируемой системы в более сложных внутренних состояниях. Однако, если проведение тестирования означает сравнение получаемых значений с эталонными, узким местом становится производительность программных симуляторов, обычно используемых для получения требуемых эталонных значений. В качестве решения можно предложить следующие пути: использование симуляторов, осуществляющих моделирование на более абстрактном уровне, задействование *условно-эталонные значений,* отказ от эталонных значений путем обратных преобразований.

Начнем рассмотрение данных подходов с использования симуляторов, моделирующих поведение системы на более высоких уровнях абстракции (по сравнению с полносистемным моделированием RTL-модели системы). Так, например, в работе [18] TLM-модель тестируемой системы (Transaction-Level Modeling – моделирование на уровне транзакций) используется для высокоуровневой генерации тестов и для сравнения результатов тестирования.

При невозможности получения эталонных значений альтернативой является использование условно-эталонных значений – значений, корректность которых не подтверждена, но которые позволяют сигнализировать о наличии расхождения с текущими результатами тестирования. Одним из часто используемых способов получения условно-эталонных значений является многократный прогон одних и тех же тестов (или модифицированных

эквивалентным образом) и принятие одного из результатов таких прогонов за условно-эталонное значение. Так, например, в работе [19] в качестве условно-эталонных значений используются значения, полученные при первом прогоне тестов.

Также возможно построение тестов, не требующих наличия эталонных значений, с использованием инструкций «прямой» и «обратной» обработки данных, как в подходе Reversi [20]. Суть метода заключается в том, что в случае корректного поведения системы исходные данные для теста и результирующие, полученные в результате обратных преобразований, будут совпадать.

2.4 Генерация тестовых данных

При онлайн-тестировании микропроцессоров тесты представляют собой последовательности тестовых программ, состоящих из отдельных инструкций, нацеленных на повышение достигнутого уровня покрытия. Тестовые программы могут включать в себя дополнительные инструкции подготовки тестового воздействия (пролог) и инструкции непосредственной проверки результата обработки тела тестовой программы (эпилог). Наиболее простой и часто используемый способ генерации тестовых последовательностей – случайный выбор тестовых инструкций, что, однако, не дает быстрого достижения приемлемого уровня полноты тестирования. С целью нахождения ошибок, возникающих в труднодостижимых внутренних состояниях, необходимо каким-то образом нацеливать генератор на определенную функциональность тестируемого микропроцессора путем выбора специальных последовательностей инструкций с подходящими параметрами.

Сначала рассмотрим некоторые более простые в своей организации способы генерации тестовых последовательностей:

1) генерация псевдослучайных последовательностей на основе сдвиговых регистров (linear-feedback shift registers) по аналогии с производственным тестированием, как это описывается в работе [21];
2) использование в качестве тестов компилируемых С-программ [22];
3) нацеливание тестов на покрытие ограничений (constraints) SystemVerilog [23] исходной RTL-модели микропроцессора;
4) присваивание вероятности выбора очередной инструкции на основе некоторых эвристик [24].

Более сложными методами организации работы генератора являются: *исполнение эквивалентных программ* и «*лакмусовые тесты*».

В подходе исполнения эквивалентных программ (Equivalent Program Execution, EPEX) [25] предлагается строить альтернативную тестовую программу P’ для тестовой программы P с помощью SMT-решателей и сравнивать состояние тестируемого микропроцессора после подачи программы P и программы P’. Подход напоминает S-QED, рассмотренный в разделе 2.1, однако применим для более широкого класса тестовых программ.

С целью повышения уровня покрытия можно также предложить использовать тесты, нацеленные на подсистему памяти (многоядерных) микропроцессоров: для проверки ограничений консистентности памяти можно использовать подход генерации «лакмусовых тестов» (litmus tests) [26], состоящих из небольшого числа инструкций, создающих особые ситуации в работе памяти. Создание тестов в упомянутой работе осуществляется на основе аксиоматической модели памяти микропроцессора, которая работает существенно быстрее обычных операционных моделей памяти.

Подытоживая, можно сказать, что существует несколько различных методов генерации тестовых данных, часть из которых позволяет получить тесты быстро, но с достижением определенного уровня покрытия. Одновременно можно разрабатывать усложненные

генераторы с помощью нацеленных методов для повышения достигнутого покрытия до приемлемого уровня.

2.5 Применимость к многопоточным микропроцессорам

Поддержка многопоточных микропроцессоров в генераторах тестовых программ для онлайн-тестирования обычно включает загрузку теста на каждое ядро и запуск их таким образом, чтобы максимизировать обращения в общую для всех процессов/ядер память [19].

Приведем некоторые работы, касающиеся вопросов тестирования памяти в многоядерных системах. В статье [27] предлагается метод оценки качества проводимого тестирования общей кэш-памяти инструментом Dacota, который сохраняет трассу запросов в кэш-память от каждого ядра, а также имеет возможность проверки корректности порядка операций работы с памятью. В статье [28] предлагается метод проверки ограничений консистентности памяти многоядерных микропроцессоров. Для этого предлагается генерация случайных тестов на основе разрешения ограничений, которые инструментируются для возможности обнаружения результата тестирования: для каждого теста генерируется набор компактных сигнатур, отражающих уникальные паттерны упорядочивания инструкций запросов в память, которые могут быть обнаружены даже после многократного исполнения теста. Уникальность и наблюдаемость сигнатур позволяют качественно оценить покрытие тестируемой памяти различными вариантами доступа.

3. Существующие инструменты онлайн тестирования

Разработка генераторов для проведения онлайн-тестирования обычно выполняется в рамках процесса разработки микропроцессоров, в котором также имеется стадия традиционной верификации. Более того, позволить себе проводить такую верификацию могут лишь достаточно крупные разработчики микропроцессоров, остальные ограничиваются традиционными средствами. Поэтому неудивительным выглядит факт, что все существующие на данный момент разработки в области онлайн-генераторов тестовых программ носят коммерческий характер. Далее мы рассмотрим инструменты онлайн-генерации, разрабатываемые ведущими поставщиками микропроцессоров: Threadmill от компании IBM [19, 29], RIS от компании ARM [30] и RITG от компании Intel [31, 32].

3.1. Threadmill

Инструмент Threadmill разрабатывается в компании IBM и используется для верификации микропроцессоров с архитектурой POWER. Инструмент возник в рамках продолжения работы над инструментом до-производственного тестирования Genesys-Pro [33] и использует его информацию о тестируемой архитектуре. Он ориентирован на тестирование многопоточных многоядерных микропроцессоров. Для генерации тестов инструментом компонуется программа-генератор на основе библиотечных средств, которая затем компилируется под целевую архитектуру и загружается на тестируемый микропроцессор. В рамках данного подхода сконфигурированный двоичный образ принято называть *исполнителем тестов* (*exerciser image*), а механизм его создания – *построителем* (*builder*). Исполнитель тестов включает базовые сервисы операционной системы, тестовые шаблоны, информацию об особенностях микроархитектуры тестируемой системы, модель архитектуры и некоторые тестовые данные (например, набор тестовых данных для арифметики с плавающей точкой). Исполнитель обеспечивает выполнение в бесконечном режиме следующих действий:

1) генерация тестовых программ на основе шаблонов, соответствующих целям тестирования, и информации об архитектуре тестируемого микропроцессора;

2) исполнение тестовых программ;
3) проверка результатов.

Далее на примере Threadmill будут рассмотрены основные проблемы проведения онлайн-тестирования.

3.1.1 Методика оценки корректности и эталонные значения

Характерной особенностью метода построения исполнителя является отсутствие в нем эталонной модели тестируемого микропроцессора. Это сделано для ускорения работы исполнителя, так как моделирование деталей работы тестируемой системы снизило бы объем вычислительных ресурсов, доступных для непосредственного процесса генерации новых тестов и их запуска. Отчасти это также связано с возможной нестабильностью работы целевой системы, что может привести к некорректным вычислениям по эталонной модели. Исполнитель тестов (см. рис. 1) состоит из следующих компонентов: спецификации (а именно, шаблонов тестовых программ и некоторой микроархитектурной информации) и компонентов, обеспечивающих генерацию тестовых последовательностей, их выполнение и проверку результатов их работы.

Рис. 1. Архитектура исполнителя тестов Threadmill
Fig. 1. Architecture of Threadmill's Test Executor

Из-за отсутствия эталонной модели возникают следующие две трудности проведения тестирования: необходимость получения эталонных значений некоторым альтернативным стандартному подходу способом, а также организация вычисления адресов инструкций, следующих за инструкциями условного перехода. Для решения первой проблемы, а именно получения значений, используемых при проверке корректности результатов, в инструменте используется подход многократного исполнения тестовых программ, однако размещенных по разным адресам. Причем заметим, что даже один и тот же адрес размещения многократно исполняемой тестовой программы не гарантирует одинаковых условий исполнения: нельзя обеспечить одинаковые начальные значения системных регистров, при отсутствии возможности их определения по эталонной модели. Для решения второй проблемы, связанной с вычислением адресов перехода, задействуется механизм исключений: в тех местах, где необходимо осуществить переход на следующий блок кода в зависимости от результата выполнения инструкции условного перехода (факт осуществления перехода нельзя определить в момент генерации теста, поскольку отсутствие эталонной модели не позволяет определить значение аргументов инструкции), в код тестовой программы вставляется инструкция, вызывающая прерывание. В обработчике прерывания происходит обращение к генератору тестовых программ, который опираясь на текущие наблюдаемые

значения системных регистров генерирует корректную инструкцию, осуществляющую переход по желаемому адресу.

3.1.2 Многоядерность и многопоточность

Ориентация генератора тестовых программ на многопоточные и многоядерные микропроцессоры означает необходимость запуска копий образа исполнителя тестов на каждом ядре и их синхронизацию друг с другом для создания разнообразных ситуаций одновременного доступа к общим ресурсам (в частности, оперативной памяти и когерентной кэш-памяти). В Threadmill для задачи синхронизации используется рассылка одновременно работающим исполнителям тестов инициализационного числа (*seed*), которое необходимо для определения их дальнейших действий. Далее исполнители тестов взаимодействуют между собой на уровне одноранговой сети.

3.1.4 Работа с исключениями

В случае Threadmill исключения используются для замены адреса на динамически вычисленные (в зависимости от расположения) в сгенерированном ассемблерном коде.

3.1.5 Выводы

Инструмент Threadmill является зрелым инструментом пост-производственной верификации микропроцессоров фирмы IBM, ориентированным на многопоточное тестирование. Инструмент связан с инструментом до-производственного тестирования Genesys-Pro, используя его информацию о тестируемой архитектуре. Основным недостатком подхода является как ограниченность способов проверки корректности, так и в целом закрытость инструмента.

3.2 RIS (MEMRIS)

В ARM разрабатывается генератор случайных тестов RIS (Random Instruction Sequences) [30], предназначенный для тестирования подсистемы памяти микропроцессоров с архитектурой ARM. С технологической точки зрения, подход, лежащий в его основе, аналогичен подходу, применяемому в Threadmill: генератор является двоичным образом, загружаемым на тестируемый микропроцессор, включает в себя информацию о целевой архитектуре, и производит оценку корректности на основе условно-эталонных значений.

Генератор, загружаемый на тестируемый микропроцессор, обеспечивает непрерывную генерацию и исполнение тестовых примеров на основе конфигурационных файлов. Конфигурационные файлы включают в себя информацию об используемых регистрах микропроцессора, свойствах областей памяти и регистров, веса инструкций для генерации тестов, количество инструкций в тесте, количество тестов и число запусков каждого сгенерированного теста. Генерация направленных тестов осуществляется за счет внедрения функций, описываемых на языке C++, вызов которых будет интегрирован генератором в последовательность инструкций случайного теста. Поскольку вызов функции имеет дополнительные накладные расходы, связанные с сохранением состояния процессора, инструмент поддерживает альтернативный механизм описания желаемых последовательностей инструкций при помощи специальных макросов, которые очень близки по внешнему виду с ассемблером ARM. Для проверки корректности результатов тестирования тестовые примеры исполняются дважды. Инструмент ориентирован на тестирование многоядерных процессоров и имеет встроенные механизмы проверки общего доступа в память, для чего включает специализированный алгоритм расположения данных в общей памяти.

Схема проведения верификации с помощью RIS выглядит следующим образом. Сначала создается тестовый план (test plan) на основе сценария тестирования. Тестовый план является отправной точкой для разработки конфигурационных файлов для настройки RIS-генератора. Сгенерированные тестовые программы подаются на тестируемое устройство для обнаружения ошибок его реализации.

Как и Treadmill, данный подход является коммерческим, и его исходный код закрыт.

3.3. RITG

Еще одним коммерческим инструментом пост-производственного тестирования микропроцессоров является *RITG* (*Random Instruction Test Generator*) [32], разработанный и используемый в компании Intel. Данный инструмент генерирует и исполняет случайные последовательности команд. Цели генерации задаются при помощи специальных конфигурационных файлов и тестовых шаблонов (*test patterns*). Инструмент включает в себя также компонент-упаковщик, который создает компактное представление результатов для отправки на инструментальную машину с запущенным симулятором тестируемой архитектуры для проверки корректности результатов. Отдельные копии RITG работают под наблюдением компонента, называемого *Лабораторным менеджером* (*Laboratory manager*), который контролирует: типы создаваемых генераторами тестов, связь тестируемых систем с тестовыми генераторами, мониторинг возникающих ошибок, перезапуск и воспроизведение тестов. Каждая отдельная тестовая последовательность маркируется уникальным индексом, который позволяет повторять конкретный тест в случае нахождения в нем ошибки. Модель памяти для загрузки ядра генератора включает в себя следующие элементы: обработчик исключений, множество тестов в двоичном виде, размеченное пространство для результатов тестов. RITG генерирует множество случайных тестов, каждый из которых начинается со сброса (reset) и включает:

1) уникальную в зависимости от инициализационного числа (seed) мини-операционную систему с высоким уровнем случайности параметров компонентов;
2) несколько тысяч случайных инструкций пользовательского кода;
3) уникальную в зависимости от инициализационного числа (seed) настройку внешних интерфейсов и специальную обвязку для проведения верификации.

Подытоживая, можно отметить, что инструмент RITG является сильно автоматизированным генератором случайных тестовых последовательностей, ориентированным на архитектуру микропроцессоров фирмы Intel, использующим инструментальную машину для оценки корректности наблюдаемого поведения.

3.4 Промежуточные выводы

Общепринятым подходом к пост-производственному тестированию микропроцессоров является использование инструментов онлайн-генерации тестовых программ. Такие генераторы загружаются непосредственно в память, доступную микропроцессору, и осуществляют его тестирование путем порождения и исполнения разнообразных последовательностей команд. Построение команд осуществляется на основе некоторого вида шаблонов, задающих их структурные и поведенческие свойства.

Все рассмотренные инструменты (см. табл. 1) являются коммерческими и закрытыми. По этой причине их функциональные возможности и характеристики сложно точно оценить. Общий недостаток данных инструментов заключается в том, что они применимы для тестирования микропроцессоров со строго определенной архитектурой. Поэтому актуальной видится задача разработки методов, позволяющих создавать инструменты онлайн генерации, применимые для широкого спектра микропроцессорных архитектур.

Из сведений, находящихся в табл. 1, можно сделать следующие выводы:

1) в части спецификаций инструменты пост-производственной верификации обычно имеют связь с инструментами до-производственного тестирования;
2) для генерации используется набор тестовых сценариев, конфигурационных файлов, соответствующих целям тестирования, и описываемых на некотором макроязыке;
3) для оценки результатов тестирования обычно используется неоднократный прогон, либо инструментальная машина с запущенным симулятором;
4) тестовым покрытием считается покрытие сценариев тестирования;
5) многопоточное тестирование подразумевает тиражирование загружаемого двоичного образа с генератором, который нацеливается на создание ситуации одновременного доступа к памяти, а синхронизация между генераторами осуществляется либо на их уровне, либо с помощью отдельного компонента управления.

Табл. 1. Рассмотренные инструменты тестирования
Table 1. Testing Toolkits Reviewed

Название инстру-мента	Спецификация и настройка генерации	Проверка корректности	Оценка полноты тестирования	Поддержка многопоточности
Threadmill	Повторно используются спецификации из инструмента Genesys-Pro. Для генерации используются сценарии тестирования.	Многократ-ный прогон одних и тех же тестов, размещенных по разным адресам	Покрытие сценариев тестирования	Одноранговое взаимодействие единых двоичных образов, настроенных с помощью инициа-лизационного числа
RIS	Спецификация явно не описывается, однако тестовые сценарии описываются на языке, близком к ассемблеру ARM. Для генерации используются сценарии (планы) тестирования.	Двукратный прогон	Покрытие сценариев тестирования, инструкций и их комбинаций	Одноранговый единый двоичный образ и использование инициализационного числа
RITG	Спецификация содержится на инструментальной машине в виде симулятора целевой архитектуры. Для генерации используются тестовые шаблоны и конфигурационные файлы.	Инструмен-тальная машина с запущенным симулятором	Покрытие сценариев тестирования	Используется Лабораторный менеджер для запуска и синхронизации отдельных копий генераторов

4. Поиск перспективного решения

Итак, как уже было отмечено, основным недостатком существующих зрелых решений пост-производственной верификации является закрытость инструментов и их ориентация на определенную архитектуру микропроцессоров. При этом существует ряд методов осуществления тех или иных аспектов верификации, рассмотренных в разделе 2, но не претендующих на полное решение.

Целью данного раздела является поиск перспективного решения пост-производственной верификации, которое должно быть открытым и должно учитывать сильные стороны перечисленных в разделе 3 инструментов. А это значит, что:

1) должна обеспечиваться высокая скорость генерации тестов для разных целевых архитектур;
2) в подходе необходима реализация оценки корректности наблюдаемого поведения либо путем многократного прогона тестов (возможно, видоизмененных), либо использованием инструментальной машины;
3) должна обеспечиваться диагностика ошибочного поведения.

Разработку перспективного инструмента предлагается вести на основе среды разработки генераторов тестовых программ MicroTESK [34], исходный код которой открыт. По аналогии со связкой инструментов Genesys-Pro и Threadmill предлагается добиться большой степени повторного использования компонентов, разработанных в рамках генератора тестов для функциональной верификации RTL-моделей микропроцессоров. Так, предлагается повторно использовать спецификации системы команд на языке nML: они включают в себя двоичное представление каждой команды, поэтому отсутствует необходимость запускать компилятор на исполняемом микропроцессоре. Аналогичным образом предлагается повторно использовать шаблоны тестовых программ на языке Ruby. Однако для непосредственного запуска генерации тестов эти шаблоны должны быть транслированы в более простое представление. Предполагается, что спецификации и шаблоны подаются на вход среды, создающей программу в двоичном коде, поддерживаемом целевым микропроцессором, которая включает в себя как возможности *генерации*, *запуска* и *проверки* тестовых программ, так и необходимые сервисы, обычно предоставляемые операционной системой.

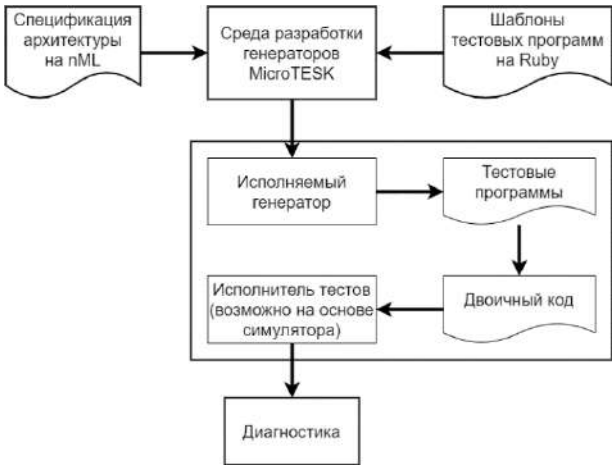


Рис. 2. Предлагаемый подход к онлайн генерации
Fig. 2. The suggested approach to online generation

Предлагаемое архитектурное решение изображено на рис. 2. Исполняемый генератор, вызываемый при работе тестовой системы на целевом микропроцессоре, создает тестовые программы, которые необходимо разместить в памяти для получения готового к запуску двоичного кода. *Исполнитель тестов* размещает эти программы, передает управление на соответствующие адреса и оценивает корректность результатов работы тестовых программ на основе какой-либо методики. Простейший вариант такой методики заключается в повторном многократном запуске одного и того же кода, где между различными случайными инструкциями вставляются инструкции, которые при их корректной работе не должны приводить к каким-либо изменениям в наблюдаемом поведении (т.е. nop, которых в микропроцессорах гораздо больше одного). Если же наблюдаемое поведение изменяется от прогона к прогону тестов, то фиксируется ошибочное поведение.

Более сложным вариантом является создание *мутантов*, т.е. таких тестовых последовательностей, которые были бы похожи на исходные, были им функционально эквивалентны и при этом имели определенные не влияющие на поведение отличия, например, измененные номера регистров. Заметим, что рассмотренный первым простейший случай является частным случаем создания мутанта. Наконец, исполнитель тестов может самостоятельно вычислять эталонные значения с помощью симулятора, созданного, например, на основе тех же nML-спецификаций. Во всех трех случаях при обнаружении ошибок сигнатуры инструкций исполненных тестовых последовательностей передаются в компонент *диагностики*, который располагается на инструментальной ЭВМ, где для анализа ошибочного поведения может быть задействован тот же MicroTESK.

Для адаптации MicroTESK к пост-производственной верификации необходимо решить еще несколько технических моментов:

1) поддержать обработку исключений, диагностика которых в пост-производственном случае крайне затруднена;
2) поддержать подсистему работы с памятью;
3) иметь поддержку многоядерности и многопоточности.

Отвечая на эти вопросы, отметим, что для каждого из них существуют как простые решения, так и более сложные, позволяющие проводить более детальные исследования целевого микропроцессора. При этом открытость инструмента тестирования позволяет при необходимости разрабатывать и добавлять отсутствующие в нем возможности.

Простыми решениями могли бы, например, быть:

1) использование обработчика исключений, возвращающего исполнение на следующую инструкцию, за той, что привела к исключению;
2) использование неоптимизированного по занимаемой памяти компонента распределения областей виртуальной памяти, однако учитывающего информацию о целевых сегментах и их отображение на физическую память;
3) создание одной и той же программы для всех ядер (потоков) с путем исполнения этой программы, определяемым номером исполняющего ее ядра, как это и делается в MicroTESK для до-производственного случая.

5. Заключение

В данной работе были рассмотрены различные подходы к созданию пост-производственных генераторов тестовых программ для микропроцессоров. Из их анализа следует, что на данный момент не существует такого решения, которое одновременно полностью бы удовлетворяло следующим выбранным критериям: скорость работы, открытый исходный код, возможность использования спецификаций, разработанных для генерации тестовых программ на этапе разработки RTL-моделей, поддержка многоядерных микропроцессоров. При этом перспективной видится как раз реализация принципа повторного использования спецификаций, так как это позволяет обрести возможность настроить генератор на разные целевые архитектуры, что повышает его ценность. Открытость исходного кода также является важным моментом.

Таким образом, в качестве основы для создания онлайн-генератора может быть использована среда генерации тестовых программ MicroTESK, которая включает формальные спецификации тестируемой архитектуры микропроцессора на языке nML, допускающие повторное использование в пост-производственном случае. В качестве метода генерации предлагается использовать генерацию случайных тестовых последовательностей на начальном этапе, а затем – шаблоны тестов по аналогии с до-производственным случаем. Методом оценки корректности может служить как многократный запуск функционально

эквивалентных тестов, так и сравнение с результатами работы симулятора. Для анализа ошибок предлагается использовать сохранение сигнатур инструкций тестовых последовательностей, приведших к наблюдаемому ошибочному поведению, для запуска аналогичных последовательностей в рамках среды MicroTESK, развернутой на инструментальной ЭВМ.

Список литературы / References

[1]. H. Foster. Trends in functional verification: a 2014 industry study. In Proc. of the Design Automation Conference, 2015, pp. 1-6.
[2]. P. Mishra, F. Farahmandi. Post-Silicon Validation and Debug. Springer, 2019, 394 p.
[3]. S. Mitra, S. A. Seshia, N. Nicolici. Post-Silicon Validation Opportunities, Challenges and Recent Advances. In Proc. of the Design Automation Conference, 2010, pp. 12-17.
[4]. Q. Xu, X. Liu. On signal tracing in post-silicon validation. Proceedings of Asia and South Pacific Design Automation Conference, 2010, pp. 262–267. DOI: 10.1109/ASPDAC.2010.5419883
[5]. B. Kumar, A. Jindal et al. A Methodology for Trace Signal Selection to Improve Error Detection in Post-Silicon Validation. In Proc. of the International Conference on VLSI Design and International Conference on Embedded Systems, 2017, pp. 147-152.
[6]. K. Basu, P. Mishra. RATS: Restoration-Aware Trace Signal Selection for Post-Silicon Validation. IEEE Transactions on Very Large Scale Integration Systems, vol. 21, issue 4, 2013, pp. 605-613.
[7]. S. B. Park, S. Mitra. Post-silicon bug localization for processors using IFRA. Communications of the ACM, vol. 53, issue 2, 2010, pp. 106-113.
[8]. S. Chen, M. Hsiao et al. A configurable bus-tracer for error reproduction in post-silicon validation. In Proc. of the International Symposium on VLSI Design, Automation, and Test, 2013, pp. 1-4.
[9]. E. El Mandouh, A. G. Wassal. Application of Machine Learning Techniques in Post-Silicon Debugging and Bug Localization. Journal of Electronic Testing, vol. 34, issue 2, 2018, pp. 163-181.
[10]. E. Singh, C. Barrett, S. Mitra. E-QED: Electrical Bug Localization During Post-silicon Validation Enabled by Quick Error Detection and Formal Methods. In Proc. of the International Conference on Computer-Aided Verification, 2017, pp. 104-125.
[11]. E. Singh, D. Lin et al. Logic Bug Detection and Localization Using Symbolic Quick Error Detection. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018, pp. 1-14.
[12]. S. Ma, D. Pal et al. Can't see the forest for the trees: State restoration's limitations in post-silicon trace signal selection. In Proc. of the International Conference on Computer-Aided Design (ICCAD), 2015, pp. 1-8.
[13]. S. Aslan, G. K. R. Chandrai, V. Siddaiah. Unified Coverage Methodology for SoC Post-Silicon Validation. Optics and Photonics Journal, vol. 6, issue 10, 2016, pp. 261-268.
[14]. B. Kumar, K. Basu et al. RTL level trace signal selection and coverage estimation during post-silicon validation. In Proc. of the International High Level Design Validation and Test Workshop, 2017, pp. 59-66.
[15]. E. E. Mandouh, A. Gamal et al. Construction of coverage data for post-silicon validation using big data techniques. In Proc. of the International Conference on Electronics, Circuits and Systems, 2017, pp. 46-49.
[16]. F. Farahmandi, R. Morad et al. Cost-effective analysis of post-silicon functional coverage events. In Proc. of the Design, Automation and Test in Europe Conference and Exhibition, 2017, pp. 392-397.
[17]. F. Farahmandi, P. Mishra. Utilization of Debug Infrastructure for Post-Silicon Coverage Analysis. In: Post-Silicon Validation and Debug. Springer, 2019, pp. 307-321.
[18]. F. Farahmandi, P. Mishra, S. Ray. Exploiting transaction level models for observability-aware post-silicon test generation. In Proc. of the Design, Automation and Test in Europe Conference and Exhibition, 2016, pp. 1477-1480.
[19]. A. Adir, M. Golubev et al. Threadmill: A post-silicon exerciser for multi-threaded processors. In Proc. of the Design Automation Conference, 2011, pp. 860-865.
[20]. I. Wagner, V. Bertacco. Post-Silicon and Runtime Verification for Modern Processors. Springer, 2011, 241 p.
[21]. N. Nicolici. On-Chip Stimuli Generation for Post-Silicon Validation. In Proc. of the International High Level Design Validation and Test Workshop, 2012, pp. 108-109.

[22]. P. Moharikar, J. Guddeti. Automated test generation for post silicon microcontroller validation. In Proc. of the International High Level Design Validation and Test Workshop, 2017, pp. 45-52.
[23]. X. Shi. Constrained-Random Stimuli Generation for Post-Silicon Validation. PhD Thesis, McMaster University, Hamilton, Ontario, Canada, 2016, 171 p.
[24]. V.M. Suryasarman, S. Biswas, A. Sahu. Automation of Test Program Synthesis for Processor Post-silicon Validation. Journal of Electronic Testing, vol. 34, 2018, pp. 83-103.
[25]. L. Klemmer, D. Große. EPEX: Processor Verification by Equivalent Program Execution. In Proc. of the Great Lakes Symposium on VLSI, 2021, pp. 33-38.
[26]. J. Alglave, L. Maranget, M. Tautschnig. Herding Cats: Modelling, Simulation, Testing, and Data Mining for Weak Memory. ACM Transactions on Programming Languages and Systems, vol. 36, issue 2, 2014, article 7, pp. 1-74.
[27]. A. DeOrio, I. Wagner, V. Bertacco. Dacota: Post-silicon validation of the memory subsystem in multi-core designs. In Proc. of the International Symposium on High Performance Computer Architecture, 2009, pp. 405-416.
[28]. D. Lee, V. Bertacco. MTraceCheck: Validating non-deterministic behavior of memory consistency models in post-silicon validation. In Proc. of the Annual International Symposium on Computer Architecture, 2017, pp. 201-213.
[29]. A. Adir, A. Nahir, A. Ziv. Concurrent Generation of Concurrent Programs for Post-Silicon Validation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 31, issue 8, 2012, pp. 1297-1302.
[30]. S. Thiruvathodi, D. Yeggina. A Random Instruction Sequence Generator for ARM Based Systems. In Proc. of the International Microprocessor Test and Verification Workshop, 2014, pp. 73-77.
[31]. B. Bentley. Validating the Intel Pentium 4 Microprocessor. In Proc. of the Design Automation Conference, 2001, pp. 244-248.
[32]. T. Bojan, F. Igor, M. Robert. Intel's Post Silicon Functional Validation Approach. In Proc. of the High Level Design Validation and Test Workshop, 2007, pp. 53-56.
[33]. A. Adir, S. Copty et al. A unified methodology for pre-silicon verification and post-silicon validation. In Proc. of the Design, Automation and Test in Europe Conference and Exhibition, 2011, pp. 1-6.
[34]. MicroTESK. URL: http://www.microtesk.org/, accessed 15.11.2021.

Информация об авторах / Information about authors

Никита Дмитриевич ЧЕРТОК – стажер-исследователь отдела технологий программирования ИСП РАН, обучается в аспирантуре ИСП РАН. Область научных интересов: автоматизация проектирования цифровой аппаратуры.

Nikita Dmitrievich CHERTOK is a researcher at the Software Engineering Department of ISP RAS. His research interests include digital hardware design automation. Nikita is a PhD student at ISP RAS.

Михаил Михайлович ЧУПИЛКО – кандидат физико-математических наук, старший научный сотрудник отдела технологий программирования ИСП РАН, старший научный сотрудник научной лаборатории «Гетерогенные компьютерные системы» РЭУ им. Г.В. Плеханова. Область научных интересов: автоматизация проектирования цифровой аппаратуры, верификация и тестирование.

Mikhail Mikhaylovich CHUPILKO is a senior researcher at the Software Engineering Department of ISP RAS, senior researcher at the Heterogeneous Computing Systems research lab of Plekhanov RUE. His research interests include digital hardware design automation, verification and testing. Mikhail has a PhD degree in Physics and Mathematics.

DOI: 10.15514/ISPRAS-2021-33(6)-10

Unidata: открытая компонентная платформа для разработки MDM-решений

[1,2] *С.В. Кузнецов, ORCID: 0000-0001-6752-6742 <sergey@unidata-platform.ru>*
[1] *А.В. Цырюльников, ORCID: 0000-0001-7509-7888 <alexey@unidata-platform.ru>*
[2] *Д.В. Кознов, ORCID: 0000-0003-2632-3193 <d.koznov@spbu.ru>*

[1] *ООО «Юнидата»,*
197110, Россия, г. Санкт-Петербург, ул. Красного Курсанта 25Б
[2] *Санкт-Петербургский государственный университет,*
199034, Россия, Санкт-Петербург, Университетская набережная 7/9

Аннотация. Управление мастер-данными (Master Data Management, MDM) является важной дисциплиной управления данными в современных компаниях. В настоящее время индустриальные решения в этой сфере активно развиваются, о чем свидетельствует, в частности, ежегодные обзоры консалтинговой компании Gartner. Однако проблема эффективной настройки стандартных MDM-продуктов под индивидуальные нужды организаций стоит очень остро. Естественным решением этой проблемы является компонентная архитектура стандартного продукта, а также его открытость. Однако, фактически, единственным индустриальным открытым компонентным MDM-продуктом является на сегодняшний день платформа Unidata, разработанная в российской компании ООО «Юнидата». В данной работе представлена архитектура этой платформы, а также проведён обзор имеющихся на рынке открытых компонентных разработок в области MDM.

Ключевые слова: корпоративные информационные системы; мастер-данные; открытые системы; компонентные системы

Для цитирования: Кузнецов С.В., Цырюльников А.В., Кознов Д.В. Unidata: открытая компонентная платформа для разработки MDM-решений. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 149-160. DOI: 10.15514/ISPRAS–2021–33(6)–10

Unidata: open source component platform for Master Data Management

[1,2] *S.V.Kuznetsov, ORCID: 0000-0001-6752-6742 <sergey@unidata-platform.ru>*
[1] *A.V. Tsyryulnikov, ORCID: 0000-0001-7509-7888 <alexey@unidata-platform.ru>*
[2] *D.V. Koznov, ORCID: 0000-0003-2632-3193 <d.koznov@spbu.ru>*

[1] *Unidata Ltd.*
Kursanta Str., St Petersburg, 197110, Russia
[2] *Saint Petersburg State University,*
7/9 Universitetskaya emb., St. Petersburg, 199034, Russia

Abstract. Master Data Management (MDM) is an important data management discipline in modern companies. Currently, industrial solutions in this area are actively developing, as evidenced, in particular, by the annual reviews of the consulting company Gartner. However, the problem of effectively customizing standard MDM products for the individual needs of organizations is very acute. The natural solution to this problem is the component architecture of the standard product, as well as its openness. However, in fact, the only industrial open component MDM product today is the Unidata platform, developed by the Russian company Unidata

LLC. In this paper, the architecture of this platform is presented, as well as an overview of open component developments in the field of MDM available on the market.

Keywords: enterprise applications; master data management; open source; component approach

For citation: Kuznetsov S.V., Tsyryulnikov A.V., Koznov D.V. Unidata: open source component platform for Master Data Management. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 149-160 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–10

1. Введение

Переход к цифровой экономике предъявляет особые требования к управлению данными в больших организациях: необходимо, чтобы эта дисциплина была столь же повсеместно используема, как, например, бухгалтерия [5]. В большой организации имеется особый вид данных, без которых трудно себе представить её нормальную работу. Речь идёт о юридических данных, клиентской базе, сведениях о поставщиках и контрагентах и пр. Пользователи этих данных рассчитывают на их согласованность в пределах организации. Эти данные часто дублируются в различных информационных системах организации, в связи с чем возникают разночтения и противоречия, которые порождают проблемы, задержки, коллизии, финансовые и имиджевые потери [3]. Таким образом, целесообразно консолидировать и сопровождать единую целостную версию таких данных, обеспечивая ею все информационные системы организации. Тогда эти данные называются мастер-данными (Master Data), а процесс их сопровождения – управлением мастер-данными (Master Data Management, MDM) [1, 3].

Существует обширный спектр MDM-стратегий и методов, структурированных в рамках энциклопедии DAMA-DMBOK (Data Management Body of Knowledge) [3]. Существует также большое количество готовых *MDM-продуктов* от таких известных компаний как IBM, Oracle, Informatica и др. (см., например, отчет Gartner за 2021 год [4]). Однако на практике встречается значительное количество случаев, когда требуются нестандартные решения (далее – *MDM-решений*), поскольку целевые организации, особенно крупные, имеют большое количество индивидуальных черт. Кроме того, часто MDM внедряется в контексте итеративной стратегии – это означает, что с его помощью решаются определённые задачи организации – улучшение каких-то бизнес-процессов, внедрение в деятельность организации новых бизнес-идей и т.д. [7]. Также организация может иметь специфичную ИТ-инфраструктуру, используя широкий спектр платформ и технологий, которые непросто интегрировать в рамках стандартных MDM-продуктов, и, следовательно, необходимо специальное MDM-решение. После успешного решения всех этих задач возможно продолжение реализации MDM в организации для решения других задач, а также в рамках других бизнес-сегментов.

Индивидуальные особенности крупных организаций, итеративная стратегия внедрения MDM, а также гетерогенный ИТ-ландшафт больших организаций требуют гибкого подхода к внедрению MDM. Этого можно достичь, используя компонентный подход. С его помощью удаётся использовать MDM для решения бизнес-задач компании, дополняя базовый продукт необходимой функциональностью, интегрируя его с готовыми сторонними компонентами для работы с данными – поиска, хранения, очистки данных и пр. Компонентный подход позволяет легко комплектовать итоговую поставку MDM-решения для организации только необходимыми ей возможностями, «отрезая» остальные. Компонентное решение оказывается существенно легче поместить в облако, нежели монолитный продукт, что оказывается дополнительным преимуществом в ряде бизнес-сценариях. Также компонентную систему существенно проще развивать в рамках популярного на данный момент открытого подхода (open source).

В данной статье представляется открытая компонентная платформа Unidata, разработанная и успешно применяемая для крупного бизнеса одноименной российской компанией.

Платформа разделена на уровни (пакеты), включает в себя механизм создания адаптеров для различно платформо-зависимых возможностей, а также предоставляет средства масштабирования.

Статья организована следующим образом. В разд. 2 представлен обзор существующих на настоящий момент открытых и компонентных MDM-продуктов. Разд. 3 является обзором платформы Unidata. Разд. 4-7 описывают основные пакеты платформы Unidata. В разд. 8 подводятся итоги и обозначаются направления для дальнейших исследований и разработок.

2. Обзор существующих MDM-продуктов

MDM-решение для конкретной организации реализуется на основе готового MDM-продукта, выбранного из имеющихся на рынке. С обзором самых известных MDM-продуктов можно ознакомиться в отчетах консалтингового агентства Gartner [4]. MDM-продукты реализуют типовую функциональность по работе с мастер-данными, в частности, хаб данных для консолидированного хранения и доступа мастер-данных, средства для создания метамодели мастер-данных, процедуры очистки, классификации иерархизации мастер-данных, а также средства доставки мастер-данных целевым информационным системам организации. Весь этот функционал не нужно реализовывать «с нуля», что существенно снижает риски данного MDM-проекта организации, а также повышает качество итогового решения. Вместе с тем очень существенным вопросом является гибкость базовых MDM-продуктов, т.е. то, на сколько легко их изменить и настроить под индивидуальные потребности конкретной организаций. Рассмотрим несколько ведущих MDM-продуктов с этой точки зрения.

Многолетним лидером в области MDM по версии Gartner является продукт Informatica MDM [8]. Этот продукт был создан в компании Siperian, которую в 2010 году выкупила компания Informatica. Как и многие другие системы этого класса, Informatica MDM двигается в сторону поддержки облачной инфраструктуры, но при этом более чем пятнадцатилетнее наследие не позволяет сделать это эффективно. На текущий момент Informatica MDM, хоть и является наиболее функционально богатым MDM-продуктом, но имеет многочисленные ограничения для кастомизации. В частности, продукт имеет «монолитную» архитектуру, а также обладает существенной сложностью – как внутренней (сложность архитектуры), так и внешней – сложность пользовательского функционала. Это является следствием того, что продукт во многом составлен из других готовых продуктов, появившихся после поглощения компанией Informatica других продуктовых компаний. Также отметим, что исходный код системы является закрытым.

Ещё одним яркий представителем классических MDM-продуктов является SAP MDG [9]. Так же, как и Informatica MDM, этот продукт обладает богатой функциональностью. SAP MDG реализован на основе проприетарной платформы SAP HANA, что обеспечивает хорошую производительность и упрощает поддержку, но существенно ограничивает потенциального пользователя при интеграции SAP MDG с технологиями и системами, основанными не на SAP. Также стоит отметить, что продукт имеет «монолитную» архитектуру и его исходный код является закрытым.

Стоит отметить, что многие производители MDM-продуктов заявляют о том, что их продукты являются компонентными. Однако в лучшем случае они поддерживают открытый программный интерфейс, а также средства кастомизации отдельных платформо-зависимых возможностей.

Рассмотрим, какие из существующих MDM-продуктов действительно являются компонентными и открытыми.

Talend MDM является первой полноценной попыткой сделать открытый MDM-продукт [10]. Его архитектура является «монолитной», но имеется набор публичных интерфейсов. Проект стартовал в 2014 году и просуществовал до 2019 года.

Egeria является полноценным открытым проектом и ориентирован на обмен метаданными разных вендоров в области корпоративных информационных системам [11]. В проекте уделяется внимание тематике Data Governance, имеются отдельные компоненты, относящиеся к MDM-сфере. Однако Eneria нельзя назвать полноценным MDM-продуктом.

Продукт Pimcore MDM является частично открытым – он имеет открытое ядро, которое поддерживает контейнеризацию и ориентировано на исполнение в облачной инфраструктуре [12]. Несмотря на то, что по заявлениям разработчиков продукт является 100% API-driven, его архитектура в основе является «монолитной».

Продукт AtroCore MDM имеет компонентную архитектуру, но не является открытой. Этому продукту не хватает полноценной поддержки облачной инфраструктуры и горизонтального масштабирования. Ещё одной особенностью продукта является использование языка PHP, что удобно при создании небольших Web-ориентированных расширений, но может оказаться проблемой при прохождении аудита безопасности целевых решений, созданных на его основе.

Интересной попыткой сделать полностью открытый MDM-продукт с гибкими возможностями по кастомизации является проект Fuyuko, стартовавший в 2020 году [13]. К сожалению, после первого выпуска проект не получил дальнейшего развития.

Из этого обзора очевидно, что открытые компонентные MDM-продукты являются актуальной темой исследований и разработки.

3. Обзор архитектуры платформы Unidata

Платформа UniData состоит из компонент, которые являются единицами пакетирования и распространения функциональности платформы. Компоненты, в свою очередь, состоят из сервисов, а последние – из Java-классов.

Компоненты платформы UniData распределены по четырём основным пакетам: **Platform Core**, **Storages**, **MDM**, **Extra MDM** (см. рис. 1). Этот набор пакетов имеет уровневую структуру – чем ниже находится пакет, тем в большей степени он является системным, чем выше – тем более прикладным. Коме того, «низлежащие» пакеты предоставляют сервисы для «вышележащих» и при этом редко бывает так, чтобы пакет непосредственно использовал сервисы несмежного с ним нижнего пакета.

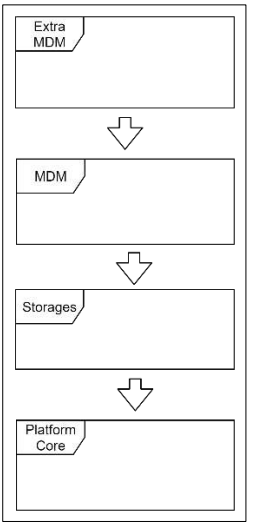


Рис. 1. Структура пакетов платформы Unidata
Fig. 1. Package structure of the Unidata platform

Кратко охарактеризуем эти пакеты.

- Пакет **Platform Core** включает в себя базовые компоненты платформы, минимально зависимые от других компонент и внешних продуктов и обеспечивающие системные сервисы платформы, такие как загрузка системы, выполнение цепочек задач, общие типы данных и пр. Компоненты этого пакета меняются крайне редко, и точно не меняются под конкретную задачу.
- Пакет **Storages** включает в себя различные хранилища данных, используемые платформой для разных функций и сервисов. Эти хранилища позволяют абстрагировать компоненты других пакетов от реальных хранилищ данных и работают с типами данных платформы (определены в Platform Core). Также большие сторонние хранилища упрощаются, так как MDM-функционал не нуждается во всех возможностях MySQL, Oracle и т.д. Платформа производит синхронизацию всех этих хранилищ на следующем уровне, в рамках пакета MDM.
- Пакет **MDM** включает компоненты, реализующие базовые функции управления мастер-данными. К этим функциям относятся управление метаданными, правила вычисления альтернативных вариантов мастер-данных, функциональность по управлению качеством данных, а также поиск дубликатов и реализация бизнес-процессов.
- Наконец, пакет **Extra MDM** включает в себя реализацию дополнительных функций управления мастер-данными, которые используются в сложных MDM-решениях.

В следующих разделах будет представлено описания каждого из пакетов. Скажем несколько слов об используемой ниже графической нотации в диаграммах, описывающих состав пакетов. В работе используется диаграмма компонент UML, но не некоторыми дополнениями.

- Зеленым цветом обозначаются компоненты, которые не меняются при создании MDM-решений на основе платформы Unidata.
- Желтым цветом обозначаются компоненты, которые существенно перерабатываются при создании MDM-решения.
- Серым цветом обозначаются сторонние компоненты, которые не являются частью платформы Unidata но используются при создании MDM-решений.

4. Пакет Platform Core

Рассмотрим пакет **Platform Core**. Компоненты этого пакета представлены на рис. 2.

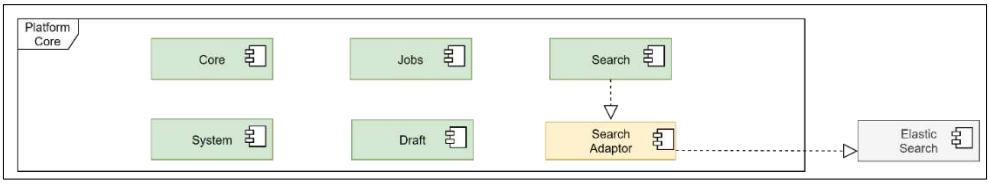


Рис. 2. Пакет Platform Core
Fig. 2. Platform Core package

Платформа и создаваемые на ее основе целевые приложения организуются в виде компонент, реализующих один стандартный интерфейс, содержащий методы для инициализации, конфигурирования, запуска и проверки. Соответственно, компонента **System** обеспечивает контроль за загрузкой всех платформы проверяя, что у загружаемых компонент имеются все необходимые компоненты для нормальной работы[1].

[1] Существуют подобные системы, например, Apache OSGi/Spring DM, Eclipse Virgo, Java 9 Jigsaw и др. Мы создали упрощённый аналог, ориентированный на нашу платформу.

Компонента **Core** содержит основные типы данных платформы, такие как специализированные перечисления, словари, динамические ссылки (с постоянной и переменной частью), специальные массивы элементов, необходимых для MDM. Также типы данных поддерживают многоязычность (минимум, русский и английский языки). Также данная компонента поддерживает дополнительные сервисы, которыми могут пользоваться другие компоненты, например, логирование, аудит, проверка лицензий и пр.

Компонента **Draft** обеспечивает работу с черновиками. Черновики является важной частью MDM, поскольку мастер-данные нуждаются в синхронизации с данными ИД/ПД, и в рамках этого процесса часто создаются временные копии данных, т.е. черновики, которые проходят различные согласования прежде чем стать «чистовиками». Также черновики могут возникать при появлении конфликтов в процессе консолидации, при выполнении процедур Data Quality и т.д. Таким образом, важно абстрагировать работу с черновиком от деталей конкретной ситуации, так как имеется существенный общий функционал: создание, удаление и поиск черновика, назначение/переназначение владельца, публикация черновика (превращение его в «чистовик») по разным стратегиям – заменить текущий «чистовик», объединить (merge) черновика с текущим «чистовиком», отказ от публикации данного черновика и пр.

В MDM имеется достаточное количество действий, которые представляют собой цепочку операций, которая должна выполняться как единое целое. Например, к хабу с мастер-данными подключается новая система, которая хочет получить имеющиеся мастер-данные. Для осуществления этого действия нужно совершить следующие операции: выполнить подключение системы к хабу, обойти хранилище мастер-данных и выбрать нужные данные, переслать эти данные системе. Компонента **Jobs** реализует интерфейс для выполнения таких действий, состоящих из цепочки операций посредством интерфейса, содержащего следующие методы: конфигурирование цепочки операций, старт, просмотр результатов, перезапуск и т.д. Поддерживается также распределённое выполнение таких цепочек в облаке.

Компонента **Search.** В MDM-задачах операция поиска используется чрезвычайно часто: например, нужно найти определенную запись на хабе данных, или нужный атрибут в метамодели, или определённый бизнес-процесс и т.д. При этом важно иметь унифицированный интерфейс для выполнения поиска и использовать существующие поисковые машины (Elasticsearch, Apache Solr, Apache Lucene); связь с конкретным используемым средством поиска осуществляется с помощью специально реализованного для этого средства варианта компоненты **Search Adapter**. Несмотря на то, что Search адаптирован для использования компонентами платформы, значительной MDM-специфики у него нет. Отметим, что при решении данной задачи не существует такого широкого спектра технологий, как, например, для хранения данных или поиска дубликатов. В силу этих причин обе компоненты располагаются, во-первых, в пакете **Platform Core**, а во-вторых, сам поиск реализован с помощью двух видов компонент, а не в четырех, как, например, для поиска дубликатов (Match), и не в трёх, как для поддержки работы с графовыми данными.

5. Пакет Storages

Теперь рассмотрим пакет **Storages.** Компоненты этого пакета представлены на рис. 3.

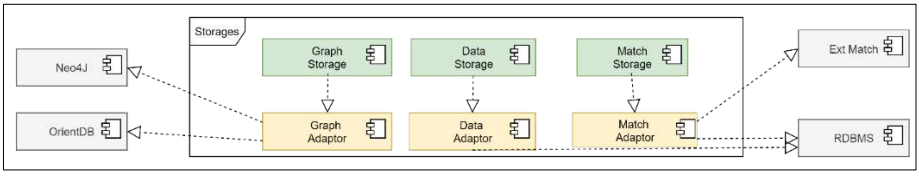


Рис. 3. Пакет Storages
Fig. 3. Package Storages

Компонента **Data Storage** представляет абстракции доступа к данным в терминах хранения (но не расчета, как компонента **Data** из пакета **MDM**). Эта компонента позволяет другим

компонентам платформы не зависеть от способа хранения мастер-данных (Oracle, MySQL, NoSQL, облако и пр. варианты). Также эта компонента реализует механизм транзакций, которые не поддерживаются, например, NoSQL.

Match Storage – компонента для абстрактного представления данных при поиске дубликатов: образца, который необходимо искать, а также кластер записей, в которых ищутся дубликаты. При этом кластер записей отличается от реальных записей: не все свойства реальных записей необходимы для поиска дубликатов, также несущественны детали хранения. Кроме этого компонента Match Storage осуществляет перевод исходного запроса в языки запросов для имеющихся готовых технологий по поиску дубликатов (matching engines), таких как Elasticsearch, Senzing и нек. др., а также в реляционные базы данных.

Graph Storage – компонента, которая абстрагирует графовую структуру данных для MDM-функционала. Графы используются в таких задачах, как каталог данных (DataCatalog) и другие специфичные для компаний сервисами, оперирующими разными иерархиями. При этом платформа использует существующие графовые хранилища, такие как Neo4J, OrientedDB и др. Таким образом, более высокоуровневые компоненты получают возможность не заботиться о деталях реализации графовых хранилищ и единообразно, в рамках всей платформы, создавать и использовать необходимые для их работы графы.

Компоненты **Data Storage Adaptor, Match Storage Adaptor** и **Graph Storage Adaptor** реализуют переходники к конкретным СУБД, средствам поиска дубликатов и графовым хранилищам, используемым в целевом MDM-решении. Для каждой из этих сторонних технологий реализуется (используется) специальный адаптер.

6. Пакет MDM

Опишем пакет MDM (рис. 4).

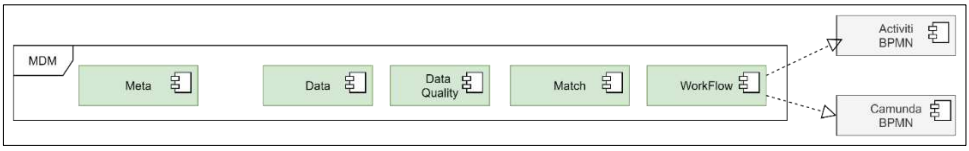


Рис. 4. Пакет MDM
Fig.4. MDM package

Компонента **Meta** предназначена для управления метамоделью (схемой, структурой) мастер-данных. Компонента позволяет создавать и редактировать типы сущностей, атрибутов и свойств атрибутов, а также виды связей. Она также поддерживает версионирование метамодели и графический интерфейс для создания и редактирования метамодели пользователем итогового MDM-решения, а также автоматически генерирует объектный программный интерфейс для доступа к мастер-данных в терминах метамодели.

Компонента **Data** отвечает за предоставление мастер-данных (в соответствии с метамоделью) другим компонентам платформы, а также для модификации мастер-данных. При этом важной чертой является функционал по расчету мастер-данных. Дело в том, что платформа хранит не просто окончательный, однозначно консолидированный вариант мастер-данных, но разные варианты для одной и той же мастер-записи. Финальный расчёт состава атрибутов и их значений производится при обращении к данным, в связи с контекстом и в соответствии с правилами. Например, могут существовать разные названия одной и той же организации – текущее и прежние; при запросе данных о юрлице выдается текущее название этой организации, а при запросе ее выполненных договоров целесообразно выдавать прежнее название, которое было у компании на момент выполнения этих договоров.

Match – компонента, которая предназначена для поиска дубликатов в некотором MDM-хранилище данных. Для этих целей не хватает компоненты Match Storage: поиск дубликатов

может производиться в хранилище мастер-данных, в черновиках, в бизнес-процессах – об особенностях таких хранилищ Match Storage не имеет информации. Более детально, компонента Match реализует следующую функциональность: задание match-правил в терминах MDM с порядком выполнения и с привязкой этих правил к различным MDM-сущностям (например, к элементам метамодели); отображение результатов выполнения Match-процедуры обратно, в термины MDM; оркестрирование поиска дубликатов, т.е. средства для задания расписания Match-процедуры и её выполнение согласно данному расписанию. Эта компонента может быть модифицирована по особенности MDM-решения, потому что может меняться подмножество, где осуществляется поиск (и так будет быстрее), а также могут появляться новые задачи.

Компонента **Data Quality** отвечает за проверку данных (поиск ошибок и определение того, что делать с ошибочными записями), а также выполняющий обогащение данных. Например, если нет названия юрлица, то такая запись вообще не допускается. А если, например, нет СНИЛС, то запись принимается, но помечается как содержащая ошибки. Вариантом обогащения может быть, например, добавление СНИЛС, извлечённого из открытых источников. Также компонента выполняет оценку интегрального качества записи, тем самым определяя степень доверия к записи.

Компонента **Workflow** поддерживает бизнес-процессы: платформа Unidata реализует подмножество стандарта BPMN 2.0 [14], включая процессы и задачи, а также события. Этот функционал важен, поскольку в рамках MDM требуется реализовывать различные бизнес-процессы, задействующие разных должностных лиц и разные департаменты, например, процесс консолидации данных. Данная компонента может использовать различные сторонние движки бизнес-процессов, такие как Activiti BPMN и Camunda BPMN.

7. Extra MDM

Наконец, перейдём к рассмотрению пакета **Extra MDM** (рис. 5).

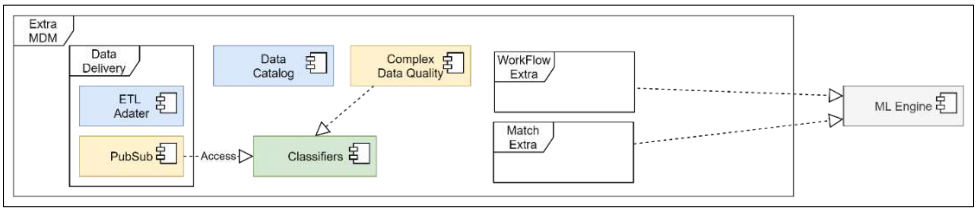


Рис. 5. Пакет Extra MDM
Fig. 5. Extra MDM package

Пакет **Data Delivery** реализует функциональность по доставке мастер-данных системам-потребителям. Компонента **PubSub** отвечает за доставку мастер-данных в режиме реального времени, пакетно и по подписке. При этом можно настроить необходимый объем данных, который требуется каждой целевой системе, а также расписание, по которому она будет получать эти данные. Также эта компонента может быть использована для закачки данных на хаб. Компонента **ETL Adapter** предназначена для передачи мастер-данных в ETL-цепочку трансформации данных[2].

Компонента **Classifiers** реализует функциональность для создания древовидных классификаторов мастер-данных, которые очень распространены в мастер-данных,

[2] ETL (Extract, Transform, Load) является обобщённой процедурой, которая состоит из следующих шагов-этапов: извлечение данных из исходных информационных систем, трансформация данных, последующая доставка данных в целевые информационные системы [15]. MDM может быть рассмотрен как один из шагов ETL-процедуры.

например, иерархия юридических лиц, иерархия видов товаров и пр. Кроме того, мастер-данные должны быть интегрированы с многочисленными внешними классификаторами. Компонента позволяет строить дерево из узлов, имеющих различные атрибуты. Также поддерживается версионирование деревьев.

MatchExtra – компонента, предоставляющая дополнительные, расширенные возможности по поиску дубликатов. Стандартная компонента **Match** базируется на понятии правила поиска дубликатов, т.е. на возможности описания правила в терминах атрибутов и свойств данных. Но в настоящее время развиваются подходы, основанные на других методах (например, машинное обучение), которые не могут быть описаны в виде правил, и стандартный подход не подходит. Для такого рода задач в платформе предполагается замена и/или расширение базовых возможностей, путем реализации дополнительных компонент. В частности, «референсной» реализацией такого подхода для решения задачи поиска дубликатов является компонента **MatchExtra**, использующая сторонний Machine Learning движок. Данная компонента не предоставляет интерфейса по заданию правил поиска дубликатов, но реализует функцию отображения результатов Match-процедуры в термины MDM. Реализация использует принцип дообучения моделей поиска дубликатов на основе решений пользователей. Данный метод желательно рассматривать как дополнительный к базовой функциональности, так как требует наличия размеченного набора данных для просторечия модели принятия решений. Что достигается путем накопления информации о решениях пользователей по дедупликации данных.

WorkflowExtra – компонента, предоставляющая дополнительные возможности для реализации сценариев по работе с матер-данными, основанных на бизнес-процессах. Эта компонента использует стандартную компоненту Workflow, но существенно расширяет её возможности за счет использования техник машинного обучения. В итоге компонента обеспечивает следующие дополнительные функции: выбор согласующего на основе предыдущей истории согласований, прогнозирование соблюдения временных регламентов при обработке запросов по работе с данными[3], поиск «совпадающих» заявок и пр.

Complex DataQulity – компонента, которая предназначена для проведения проверок мастер-данных, ориентированных на группу записей, в то время как компонента Data Quality позволяет проверять лишь одну запись. Например, проверка набора связанных записей одного или разных типов на соответствие атрибутов разных записей в наборе между собой, проверка агрегатов значений атрибутов разных записей набора на соответствие агрегата условию, или проверка соответствия значений атрибутов мастер и классификационных данных. Компонента обеспечивает дополнительный уровень представления композитного набора записей. Это представление вносит дополнительный слой агрегации изолированных моделей данных, хранящихся в платформе, и, зачастую, зависит от набора используемых компонент платформы.

Компонента **DataCatalog** реализует инвентаризированный набор знаний о видах данных компании, местах их происхождения, использования, хранения, предметных областях, и о взаимосвязях. Данная компонента нужна почти в любой крупной организации, являясь единым справочником имеющихся данных, что крайне важно для цифрового управления. Соответствующую справочную информацию нецелесообразно выяснять у авторов соответствующих ИС, поскольку таких ИС в организации может быть много (до 1000 и более).

[3] SLA (Service Level Agreement) – устоявшийся на сегодняшний день термин, определяющий скорость реагирования сервисных компаний, подсистем и пр. на запросы клиентов.

8. Заключение

В данной статье представлена открытая компонентная платформа Unidata, предназначенная для создания MDM-решений, ориентированных на обеспечение конкретных потребностей крупных бизнес-организаций. Данная платформа была создана на основе продукта Unidata и уже прошла успешную апробацию в ряде промышленных проектов. В качестве дальнейших направлений исследований и разработки можно указать доработку платформы для использования в облачных инфраструктурах, а также создание и интеграцию в архитектуру отдельных компонент, основанных на машинном обучении.

Список литературы / References

[1]. Gartner Glossary, Available at: https://www.gartner.com/en/glossary, accessed 01.12.2021.
[2]. Silvola R., Jääskelainen O. et al. Managing one master data – Challenges and preconditions. Industrial Management & Data Systems, vol. 111, issue 1, 2011, pp. 146-162.
[3]. DAMA-DMBOK: Data Management Body of Knowledge, Technics Publications, Second edition, 2017, 590 p.
[4]. Parker S., Hawker M., Walker S. Magic Quadrant for Master Data Management. Gartner 2021.
[5]. Khatri V., Brown C.V. Designing data governance. Communications of the ACM, vol. 53, issue 1, 2010, pp. 148-152.
[6]. Zmud R.W. An Examination of 'Push-Pull' Theory Applied to Process Innovation in Knowledge Work. Management Science, vol. 30, issue 6, 1984, pp. 727-738.
[7]. Кузнецов С.В., Кознов Д.В. Управление мастер-данными в рамках итеративного подхода. Онтология проектирования, том 11, no. 2, 2021, pp. 170-184.
[8]. Cloud-first, AI-powered Master Data Management. Available at: https://www.informatica.com/products/master-data-management.html, accessed 01.12.2021.
[9]. SAP Master Data Governance. Available at: https://www.sap.com/cis/products/master-data-governance.html, accessed 01.12.2021.
[10]. Clean, complete, uncompromised data for everyone. Available at: https://www.talend.com/, accessed 01.12.2021.
[11]. Egeria. Available at: https://odpi.github.io/egeria-docs/, accessed 01.12.2021.
[12]. Pimcore. Available at: pimcore.com, accessed 01.12.2021.
[13]. Fuyuko. Available at: fuyuko.org, accessed 01.12.2021.
[14]. Business Process Model and Notation (BPMN). Version 2.0. OMG, 2011, 538 p.
[15]. Vassiliadis P., Simitsis A., Skiadopoulos S. Conceptual modeling for ETL processes. In Proc. of the 5th ACM International Workshop on Data Warehousing and OLAP, 2002, pp. 14-21.

Информация об авторах / Information about authors

Сергей Викторович КУЗНЕЦОВ, исполнительный директор Юнидата, преподаватель кафедры прикладной кибернетики СПбГУ. Научные интересы: инженерия мастер-данных, требования к мастер-дата проектам, средства реализации управления мастер-данными, технологии доставки консолидированных данных потребителям.

Sergey Viktorovich KUZNETSOV, Executive Director of Unidata, Lecturer at the department of Applied Cybernetics of SPbU. Research interests: master data engineering, requirements for master data projects, means of implementing master data management, technologies for delivering consolidated data to consumers.

Алексей Владимирович ЦЫРЮЛЬНИКОВ, главный архитектор. Научные интересы: управление мастер-данными, архитектура средств управления мастер-данными, средства разработки и архитектура enterprise-приложений.

Alexey Vladimirovich TSYRYULNIKOV, chief architect. Research interests: master data management, master data management tools architecture, development tools, and enterprise application architecture

Дмитрий Владимирович КОЗНОВ, доктор технических наук, профессор кафедры системного программирования. Научные интересы: программная инженерия, модельно-ориентированная разработка программного обеспечения, программные данные, машинное обучение.

Dmitry Vladimirovich KOZNOV, Doctor of Technical Sciences, Professor of the System Programming Department. Research interests: software engineering, model-driven software development, program data, machine learning.

DOI: 10.15514/ISPRAS-2021-33(6)-11

Система маркирования документов для проведения расследований при их утечке

[1] *Д.О. Обыденков, ORCID: 0000-0002-9296-6333 <obydenkov@ispras.ru>*
[1] *А.Ю. Якушев, ORCID: 0000-0001-6089-6505 <yakushev@ispras.ru>*
[1] *Ю.В. Маркин, ORCID: 0000-0003-1145-5118 <ustas@ispras.ru>*
[1] *С.А. Фомин, ORCID: 0000-0002-1151-2189 <fomin@ispras.ru>*
[1] *А.Е. Фролов, ORCID: 0000-0001-7616-2354 <aefrolov@ispras.ru>*
[2] *С.В. Козлов, ORCID: 0000-0003-1269-1681 <kozlov_sv@mail.ru>*
[2] *Д.Д. Громей, ORCID: 0000-0003-1066-556X <gromej@academ.msk.rsnet.ru>*
[2] *А.В. Козачок, ORCID: 0000-0002-6501-2008<a.kozachok@academ.msk.rsnet.ru>*
[3] *Б.В. Кондратьев, ORCID: 0000-0001-6348-117X <gae@mil.ru>*

[1] *Институт системного программирования РАН им. В.П. Иванникова, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Академия Федеральной службы охраны Российской Федерации, 302015, Россия, г. Орёл, ул. Приборостроительная, д. 35*
[3] *Министерство обороны Российской Федерации, 119160, г. Москва, ул. Знаменка, д. 19*

Аннотация. В статье представлена система расследования утечек конфиденциальных текстовых документов, где в качестве каналов утечек рассматриваются изображения, полученные путем сканирования/фотографирования напечатанных документов, а также фотографии документов на экране монитора. Таким образом, система нацелена на защиту каналов, не защищаемых традиционными DLP-решениями. В качестве механизма защиты документов выбрано внедрение цифрового водяного знака (ЦВЗ), предполагающее изменение визуального представления документа. В случае анонимной утечки конфиденциального документа ЦВЗ позволит установить сотрудника, допустившего утечку – намеренно или в результате нарушения протокола безопасности. В статье описана архитектура системы, состоящая из клиентской и серверной частей. На рабочие станции сотрудников устанавливаются компоненты, обеспечивающие внедрение ЦВЗ в документы, отправляемые на печать или выводимые на экран монитора. Факты маркирования документов регистрируются и отправляются на удаленный сервер, использующий данную информацию при расследовании утечек аналитиком службы безопасности. Разработаны алгоритмы маркирования для встраивания ЦВЗ в текстовые документы, выводимые на печать и экран монитора. При маркировании документа на экране монитора ЦВЗ встраивается в межстрочные интервалы: цифровая метка кодируется последовательностью затемненных и осветленных областей. Для встраивания ЦВЗ в печатаемые документы разработано три алгоритма маркирования: на основе горизонтального и вертикального смещения слов, а также посредством изменения яркости отдельных фрагментов слов. Разработана методика тестирования алгоритмов маркирования в условиях, приближенным к условиям эксплуатации, оценена область применимости алгоритмов. Проведен анализ вероятных атак на систему, и сформулирована модель нарушителя.

Ключевые слова: защита от утечек информации; цифровой водяной знак; слепое извлечение ЦВЗ; устойчивость к преобразованиям print-scan, print-cam, screen-cam; обработка изображений

Для цитирования: Обыденков Д.О., Якушев А.Ю., Маркин Ю.В., Фомин С.А., Фролов А.Е., Козлов С.В., Громей Д.Д., Козачок А.В., Кондратьев Б.В. Система маркирования документов для проведения

расследований при их утечке. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 161-174. DOI: 10.15514/ISPRAS–2021–33(6)–11

Document Marking System for Leak Investigations

[1] *D.O. Obydenkov, ORCID: 0000-0002-9296-6333 <obydenkov@ispras.ru>*
[1] *A.Yu. Yakushev, ORCID: 0000-0001-6089-6505 <yakushev@ispras.ru>*
[1] *Yu.V. Markin, ORCID: 0000-0003-1145-5118 <ustas@ispras.ru>*
[1] *A.E. Frolov, ORCID: 0000-0001-7616-2354 <aefrolov@ispras.ru>*
[1] *S.A. Fomin, ORCID: 0000-0002-1151-2189 <fomin@ispras.ru>*
[2] *S.V. Kozlov, ORCID: 0000-0003-1269-1681 <kozlov_sv@mail.ru>*
[2] *D.D. Gromey, ORCID: 0000-0003-1066-556X <gromej@academ.msk.rsnet.ru>*
[2] *A.V. Kozachok, ORCID: 0000-0002-6501-2008<a.kozachok@academ.msk.rsnet.ru>*
[3] *B.V. Kondrat'ev, ORCID: 0000-0001-6348-117X <gae@mil.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
[2] *Academy of Federal Guard Service,*
35, Priborostroitel'naya st., Orel, 302015, Russia
[3] *Ministry of Defence of the Russian Federation,*
19, Znamenka Str., Moscow, 119160, Russia

Abstract. This paper presents a confidential text documents leakage investigation system, focused on leak channels by documents printing and screen photographing. Internal intruders may print confidential document, take paper copy out of protected perimeter, make document image by scanner and perform anonymous leak. Also, intruders may take a photo of printed confidential document or displayed on workstation screen using personal mobile phone. Described leakage channels are weakly covered by traditional DLP systems that are usually used by enterprises for confidential information leak protection. Digital watermark (DWM) embedding is chosen as a document protection mechanism by implying changing of document image visual representation. In case of confidential document anonymous leak embedded DWM would enable the employee to determine what leak intentionally or by security protocol violation. System architecture consists of different type components. Employees' workstation components provide DWM embedding into documents, which are sent for printing or displayed on screen. Information about watermark embedding is sent to a remote server that aggregates marking facts and provides it to security officer during investigation. Text document marking algorithms are developed, which embed DWM into printed and displayed on screen documents. Screen watermark is embedded into interline space interval, information is encoded by sequence of lightened and darkened spaces. DWM embedding into printed documents is implemented by three algorithms: horizontal and vertical shift based, font fragments brightness changing based. Algorithms testing methodology is developed in view of the production environment, that helped to evaluate the application area of algorithms. Besides, intruder model was formulated, system security was evaluated and determined possible attack vectors.

Keywords: data leakage prevention; blind watermarking methods; print-scan, print-cam, screen-cam watermarking; image processing

For citation: Obydenkov D.O., Yakushev A.Yu., Markin Yu.V., Frolov A.E., Fomin S.A., Kozlov S.V., Gromey D.D., Kozachok A.V., Kondrat'ev B.V. Document Marking System for Leak Investigations. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 161-174 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–11

1. Введение

Проблема утечки конфиденциальных документов становится все более актуальной для организаций, работающих над цифровизацией процессов. Особую значимость в организациях представляют следующие данные:

- финансовые и операционные показатели;

- информация о технологиях и процессах;
- сведения о сотрудниках, клиентах, поставщиках.

Данная информация представляет большую ценность для конкурентов организации, и утечка подобной информации может повлечь за собой значительный финансовый и репутационный ущерб организации. Все чаще причиной утечки становятся «инсайдеры» - сотрудники организации, работающие в сговоре с нарушителями внешнего характера. По данным исследования InfoWatch [1] количественное соотношение утечек по типу внешнего и внутреннего нарушителя в мире примерно равно, но в России доля утечек из-за внутренних нарушителей достигает 79% [2].

Исследователи ИБ выделяют ряд каналов утечки информации, среди которых чаще всего используется сетевой канал (79,3% от всех утечек в 2020 году). Утечки осуществляются через сеть Интернет посредством отправки конфиденциальных документов на личную почту, облачный сервис или сетевой ресурс. Для защиты от такого рода утечек применяются DLP-системы (Data Leak Protection). Решения данного типа позволяют предотвращать утечку в режиме реального времени, поскольку осуществляют непрерывный мониторинг периметра - такие системы относят к *активным*. Однако, система может ошибаться и блокировать допустимую активность пользователей, поскольку на практике сложно избежать ложноположительных срабатываний. Поэтому в ряде случаев обосновано использование пассивных DLP-систем, нацеленных на регистрацию инцидентов потенциальных утечек информации. Обнаружение конфиденциального документа может выполняться как по формальным признакам (специальным атрибутам документа), так и на основе анализа содержимого. Зачастую для поиска по содержимому задаются нечеткие критерии соответствия, как например поиск по сигнатурам или регулярным выражениям, более продвинутые методы опираются на лингвистический анализ содержимого или вычисляют цифровой отпечаток документа. И наконец, ряд систем использует OCR (Optical Character Recognition) для поиска конфиденциальных документов [3][4].

Компоненты DLP системы размещаются в различных точках периметра и соответственно берут на себя различные функции. *Агент* размещается на рабочем месте пользователя (*data-in-use*) и охватывает сразу несколько плоскостей возможных утечек. Агент контролирует потоки данных через локальные устройства ввода-вывода: копирование файлов на внешние носители, отправка документов на печать, обмен файлами через устройства Bluetooth и другие. Контроль сетевых взаимодействий также осуществляется в данной точке, поскольку позволяет эффективно контролировать данные, передаваемые через защищенные протоколы (*data-in-motion*): web, почтовые, VoIP и протоколы мессенджеров. Компоненты DLP, размещаемые на сетевом шлюзе, контролируют содержимое сетевого трафика при помощи технологии DPI (Deep Packet Inspection). К компонентам такого типа предъявляются особые требования к производительности, поскольку пропускная способность сетей в корпоративных системах может достигать десятков гигабит. Требования к производительности сильно ограничивают сложность алгоритмов анализа, однако контроль на данном уровне чрезвычайно важен, поскольку позволяет обеспечить защиту от утечек с устройств, неподконтрольных администратору сети, например, с личных мобильных устройств или устройств, используемых в рамках концепции BYOD (Bring Your Own Device). Компоненты поиска размещаются рядом с хранилищем конфиденциальной информации (*data-at-rest*) и выполняют непрерывное сканирование ресурсов организации на предмет небезопасного размещения документа, выполняя таким образом превентивную защиту от утечек. При эксплуатации таких систем особое значение имеет генерация и визуализация отчетов о событиях, необходимые для работы аналитика службы безопасности.

Лидеры рынка DLP решений совмещают в себе несколько типов компонентов и обеспечивают комплексный контроль цифровых ресурсов организации, однако актуальной остается проблема утечек с использованием бумажных копий и личных мобильных

телефонов. Распечатанный документ выходит из-под контроля системы DLP, что позволяет потенциальному "инсайдеру" вынести бумажную копию за пределы защищаемого контура и анонимно разместить скан или фотографию конфиденциального документа в сети Интернет. Также фотографии, сделанные личным мобильным телефоном с экрана рабочего компьютера, не могут отслеживаться системами DLP. На российском рынке присутствуют решения Trace Doc [5], EveryTag [6] и SafeCopy [7], нацеленные на борьбу с утечками данного типа. Однако использованные разработчиками алгоритмы маркирования требуют для расследования утечки оригинальную версию документа, что была отправлена на печать или сфотографирована с экрана. Данное требование подразумевает создание централизованного защищенного хранилища оригинальных документов и сведений о маркировании. Соответственно, если нужная версия документа отсутствует в хранилище, то извлечение метки не представляется возможным.

В данной работе рассматривается система, нацеленная на борьбу с анонимными утечками конфиденциальных документов с использованием бумажных носителей и фотографий экрана. Основной упор делается на расследование утечки постфактум и выявление внутреннего нарушителя, осуществившего утечку или нарушившего протокол безопасности, что привело к утечке информации. Согласно выбранной концепции сотруднику службы безопасности для расследования инцидента утечки достаточно изображения документа, несанкционированным образом покинувшего защищаемый периметр. Статья организована следующим образом: в первом разделе обосновывается мотивация разработки системы, второй раздел описывает архитектуру системы, третий раздел включает краткое описание разработанных алгоритмов, в четвертом выполняется анализ разработанной системы. Система маркирования состоит из множества компонентов, подробное описание наиболее значимых компонентов системы содержится в статьях [8,12,26,27]. Данная статья описывает взаимодействие между отдельными компонентами и особенности работы системы в целом.

2. Архитектура системы

В качестве ключевого принципа функционирования системы был выбран подход встраивания *цифрового водяного знака* (ЦВЗ) в документ. В документ внедряется битовая последовательность заданной длины – цифровая метка, уникальным образом идентифицирующая распространителя. Механизм выдачи цифровых меток должен позволять однозначно определить источник утечки. Процесс *внедрения* ЦВЗ в документ обозначается в статье как *маркирование* – преобразование документа согласно определенному алгоритму маркирования. Для *извлечения* метки требуется *маркированный документ*. Процесс восстановления документа к состоянию до встраивания метки или близкое к этому обозначается как *стирание* метки. Отметим, что для извлечения или стирания цифровой метки оригинальный документ не требуется. Это одно из ключевых преимуществ разработанной системы.

Компоненты системы (рис. 1) разделяются на два класса: клиентские и серверные. Первые устанавливаются на рабочую станцию пользователя и обеспечивают маркирование печатаемых документов и документов, выводимых на экран. Встраиваемая метка генерируется односторонней криптографической функцией на основе сведений о рабочей станции и учетной записи *пользователя*. Цифровая метка встраивается в документы, отправляемые на печать или отображаемые на экране монитора рабочей станции. Информация о маркировании регистрируется и передается на удаленный сервер, агрегирующий информацию о маркировании со всех рабочих станций периметра. В дальнейшем при утечке конфиденциальной информации *аналитик службы безопасности* использует специальный web-интерфейс анализа инцидента для установления виновника утечки. Аналитик определяет пользователя, допустившего утечку, и рабочую станцию, которую он использовал.

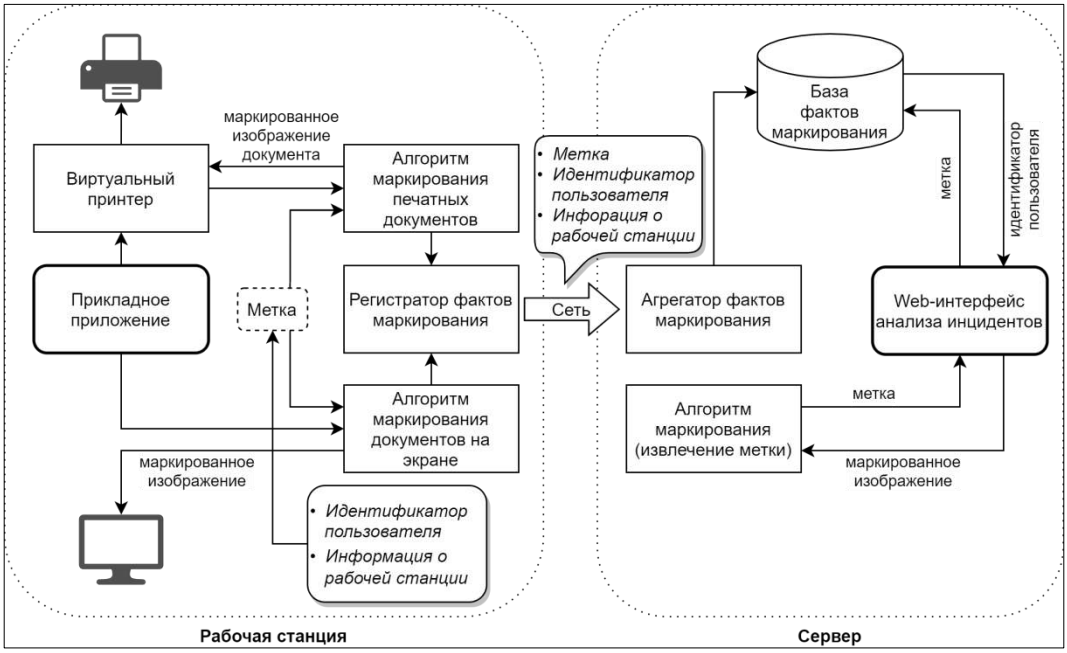


Рис. 1. Архитектура системы
Fig. 1. System architecture

Рассмотрим подробнее процессы, происходящие на рабочей станции при отправке документа на печать (рис. 2).

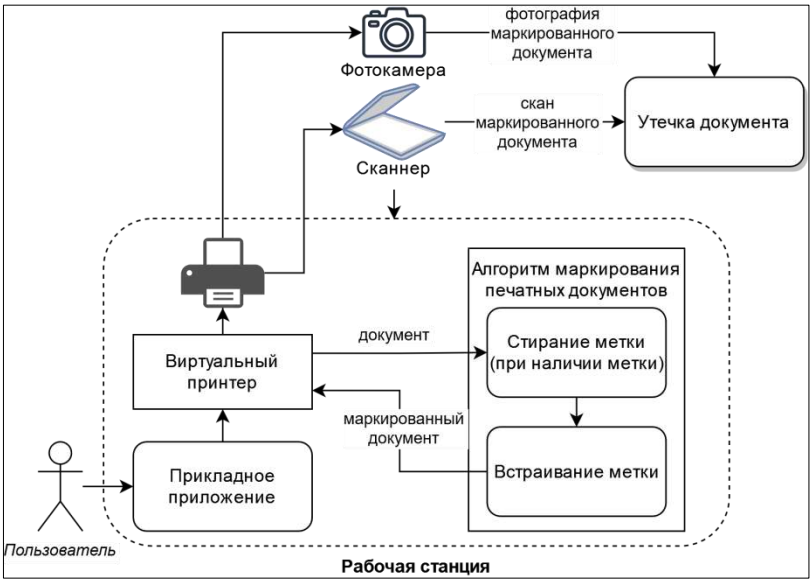


Рис. 2. Маркирование документов при печати
Fig. 2. Printed documents marking

При установке системы на рабочую станцию устанавливается и конфигурируется виртуальный принтер, который обрабатывает документы и перенаправляет их на физический принтер [8]. Виртуальный принтер берет на себя важные функции: разбивает

многостраничный документ на отдельные страницы, растеризует каждую из них, при необходимости корректирует ориентацию страницы и маркирует каждую страницу документа посредством *алгоритма маркирования*. При печати документа из прикладного приложения, как например, Word или Adobe Reader, пользователь должен выбрать виртуальный принтер как устройство печати. Соответственно, при установке системы на рабочую станцию должны применяться политики, запрещающие печать через иные устройства помимо виртуального принтера.

Перед внедрением метки выполняется стирание ранее встроенной метки, если она присутствует. Это возможно, если пользователь отправил на печать отсканированный документ с внедренной меткой. В этом случае необходимо заменить метку в документе на метку данного пользователя. В случае успешного встраивания цифровой метки страница отправляется на печать. Если происходит ошибка или встраивание метки в данную страницу невозможно, то на печать отправляется исходная (немаркированная) страница. Стратегия полного запрета печати немаркированных документов может создать значительные сложности для пользователей, а также сфокусировать их внимание на различиях между распечатанными документами и документами, отправленными на печать. Информация о маркировании – факт – сохраняется в лог-файл и отправляется на удаленный сервер. Факт маркирования включает:

- Тип маркирования (документ выведен на печать или на экран);
- Встроенная в документ цифровая метка;
- Дата и время маркирования;
- Идентификатор пользователя;
- Информация о рабочей станции:
  - MAC-адрес;
  - IP-адрес;
  - серийный номер жесткого диска.

Рис. 3. Маркирование документов, выводимых на экран
Fig. 3. Marking of documents displayed on the screen

Схожим образом построен процесс маркирования выводимой на экран информации (рис. 3). Алгоритм маркирования экрана взаимодействует с операционной системой, от которой

получает информацию о состоянии графического интерфейса, положении и содержании окон графических приложений. На основе этой информации программа маркирования экрана генерирует ЦВЗ, который накладывается поверх всех остальных графических приложений.

Сервер получает уведомления о событиях маркирования. Для экономии дискового хранилища и оптимизации поиска при проведении расследования выполняется агрегация фактов маркирования - объединение множества одинаковых событий маркирования в одно (маркирование документа на экране) путем замены отметок времени на временной интервал. Для расследования произошедшей утечки документа за пределы периметра требуется наличие изображения маркированного документа. Аналитик загружает изображение через специальный web-интерфейс, после чего запускается программа извлечения метки из изображения. По извлеченной метке выполняется поиск в базе данных, и в качестве результата аналитику выдается идентификатор пользователя и его рабочей станции.

3. Алгоритмы маркирования

3.1. Алгоритмы маркирования документов на экране

В рамках работы над системой маркирования документов, выводимых на экран монитора, были рассмотрены существующие решения внедрения цифровой метки в изображения на экране, устойчивые к искажениям, возникающим при фотографировании экрана. Первый подход [9], предложенный в 2018 году, основан на изменении яркости областей на экране. Каждому биту цифровой метки ставится в соответствие круговая область, яркость которой повышается или понижается в зависимости от значения бита. Недостатком метода является высокая заметность круговой области, если метка встраивается в одноцветное изображение. В основу двух других методов [10, 11] положена идея поиска таких признаков областей для встраивания метки, что эти признаки сохраняются на фотографии экрана. Для первого алгоритма [10] признаки областей задаются положением особых точек алгоритма I-SIFT. Для второго метода [11] поиск осуществляется с помощью детектора Харриса-Лапласа. Цифровая метка встраивается в домен преобразования найденных областей – дискретного косинусного преобразования или дискретного преобразования Фурье. Методы, внедряющие метку в домен преобразований, обладают общим недостатком: цифровая метка незаметна на изображениях, богатых цветами, но хорошо заметна на изображениях текстовых документов, как правило, черно-белых (пример в статье [12]).

Проведенный анализ существующих подходов показал необходимость разработать собственный метод встраивания цифровой метки в изображение документа, выводимого на экран монитора. Было принято решение встраивать цифровую метку путем изменения яркости областей на экране. С целью уменьшить заметность метки на одноцветных областях метод должен изменять яркость только в тех областях на экране, на которых присутствует текст.

К разрабатываемой системе маркирования выдвигается ряд требований. Метод должен быть устойчив к искажениям, возникающим при фотографировании экрана. Цифровая метка должна быть незаметна для пользователя устройства. Метка должна встраиваться в режиме реального времени. Маркироваться должен любой документ на экране независимо от его формата – текстовый файл, изображение документа и др. Извлечение цифровой метки должно проводиться в условиях отсутствия оригинального документа.

Разработанный алгоритм внедрения метки в текстовые документы [12], выводимые на экран монитора, работает следующим образом. Цифровая метка внедряется в межстрочные интервалы текста. Метка является последовательностью светлых и темных прямоугольных областей, кодирующих биты 0 и 1 соответственно. В каждый межстрочный интервал встраивается 16 бит, поэтому для встраивания цифровой метки размером 32 бита достаточно

двух межстрочных интервалов (или трех строк текста). Отображение метки производится при помощи частично прозрачного окна-оверлея, накладываемого поверх остальных окон, открытых на рабочем столе устройства. Содержимое окна-оверлея регулярно обновляется в соответствии с расположением окон с текстом на экране так, что последовательности прямоугольных областей соответствуют межстрочным интервалам маркируемого текста.

Для работы алгоритма извлечения метки требуется только фотография документа, выведенной на экран монитора. На снимке определяются межстрочные интервалы текста. В основу алгоритма извлечения метки положен факт, что цифровая камера хорошо различает плавные переходы яркости. Переходы яркости в межстрочных интервалах на фотографии соответствуют переходам яркости в цифровой метке, внедренной на маркированный документ, что позволяет определить значения битов встроенной метки. Это позволяет алгоритму извлекать цифровую метку из фотографии без изображения оригинала документа, выведенного на экран монитора. Разработанный алгоритм не только определяет значения битов встроенного сообщения, но также дает вероятностную оценку корректности полученных значений. Извлечение цифровой метки может проводится многократно с перебором параметров с целью достичь максимального значения этой оценки.

Разработанный алгоритм извлечения был протестирован на предмет устойчивости к искажениям, возникающим при фотографировании экрана. Цифровая метка успешно извлекалась при фотографировании экрана на различных расстояниях (30 см, 50 – 100 см) и углах (0^o – 45^o) между камерой и экраном, а также при JPEG-сжатии фотографии высокой степени, вплоть до коэффициента качества JPEG 15. Цифровая метка частично извлекалась из фотографий, сделанных на расстоянии 40 см, при углах между камерой и экраном не более 60^o, а также при коэффициенте качества JPEG не менее 10.

3.2. Алгоритмы маркирования документов при печати

Задача маркирования документов при печати широко освещена во множестве публикаций. Существует большое разнообразие методов внедрения метки, опирающихся как на область преобразований (*transform domain*), так и на пространственную область (*spatial domain*). К первым относятся подходы на основе дискретного преобразования Фурье (DFT) [13], дискретно-косинусного (DCT) [14] или вейвлет преобразований (DWT). Вторая группа методов предполагает модификацию оригинальных документов с использованием информации о структуре, например, данные о местоположении слов, строк или текстовое содержимое. Эту группу можно разделить на подгруппу лингвистических методов, изменяющих семантические и/или синтаксические свойства текстового содержимого документа, а также структурные методы, изменяющие параметры визуального представления документа, но не изменяющие смысл/содержимое текста [15]. Семантические методы могут корректировать правописание, заменять слова на синонимы и аббревиатуры [16, 17]. Синтаксические методы существенно не изменяют смысл текста, а используют его свойства и особенности, как например, заменяют символы букв одного алфавита на визуально схожие символы букв другого алфавита [18]. Одни из самых ранних работ по маркированию документов использовали структурный подход и основывались на вертикальном смещении текстовых строк вверх или вниз [19, 20]. Позднее подход со смещением применялся для горизонтального смещения отдельных слов [21, 22]. Также было опубликовано большое количество работ, посвященных структурному кодированию (термины внедрение и кодирование ЦВЗ следует считать равнозначными). В них используются свойства форматирования текста: размер, цвет, особенности начертания шрифта и другие свойства: например, методы кодирования на основе высвечивания контуров символов [23] или искажения шрифтов [24]. Для арабских языков разработан метод, использующий особенности начертания букв при кодировании: сдвиги точек/увеличение длины черты в определенных словах [25].

По результатам анализа особенностей различных методов и техник внедрения цифровых меток в документы при печати было принято решение об использовании в качестве базового метода для решения поставленной задачи подход на основе структурного кодирования. Применение лингвистических методов невозможно, поскольку в результате внедрения изменяется содержимое документов. Подходы, изменяющие область преобразования, существенно ухудшают качество документов и делают внедренную цифровую метку заметной. Методы, предложенные в [23, 24], предъявляют значительные требования к разрешению и качеству изображений маркированных документов при извлечении, что ограничивает применение методов на практике.

В рамках выбранной схемы функционирования системы были сформулированы требования к алгоритму маркирования. Должны поддерживаться следующие операции обработки документа: внедрение, извлечение, стирание метки. Последняя не встречалась в публикациях по данной тематике. В то же время необходимость поддержки операции стирания метки существенно ограничивает возможные преобразования над документом при встраивании метки, поскольку в этом случае маркирование документа должно быть обратимым. Другим значимым ограничением является возможность извлечения метки из маркированного документа без оригинального документа. Также при разработке методов маркирования учитывались требования к работоспособности рассмотренных подходов при значительных искажениях, возникающих при многократной печати, сканировании и фотографировании документов.

Было разработано три алгоритма маркирования:

1) на основе горизонтального смещения слов [26];
2) на основе вертикального смещения слов [27];
3) на основе перечеркивания слов [27].

Первый алгоритм развивает идеи, положенные в методики кодирования ЦВЗ на основе смещения, представленные в работах [19-22]. Ключевая идея механизма кодирования – горизонтальное смещение слов. Документ разбивается на множество строк, строки посредством жадного алгоритма разбиваются на блоки последовательно расположенных в строке слов, разделенных четырьмя или двумя пробелами. При внедрении метки в документ слова горизонтально смещаются таким образом, чтобы величина одного из пробелов в блоке увеличилась, а величина остальных пробелов уменьшилась так, чтобы общая длина блока осталась неизменной. Позиция удлиненного пробела в блоке позволяет кодировать цифровую метку: для блоков из четырех пробелов – 2 бита, для блоков из двух пробелов – 1 бит. Стирание ранее внедренной метки выполняется одновременно с встраиванием метки и не требует дополнительных операций, однако, исходное положение слов в документе восстановлению не подлежит.

Рис. 4. Пример базовой линии и медианы слова
Fig. 4.Word baseline and median example

В основе второго алгоритма маркирования – вертикальное смещение слов. Заметим, что упомянутые ранее работы использовали для кодирования информации вертикально смещенные строки [19, 20] или горизонтально смещенные слова [21, 22]. Разработанный алгоритм имеет большую информационную емкость по сравнению с алгоритмами, использующими вертикальное смещение строк. Для кодирования одного бита информации слово вертикально смещается вверх или сохраняет исходного положение. Первое и последнее

слова в строке никогда не смещаются и задают «нулевой уровень» смещения, то есть уровень несмещенных слов. Извлечение метки выполняется построчно с использованием алгоритма Витерби, позволяющего определить наиболее вероятную последовательность смещений слов в строке. Относительное положение слов определяется путем сравнения их базовых линий (рис. 4). При стирании метки выполняется обратное вертикальное смещение слов.

В третьем алгоритме кодирования изменяется яркость отдельных фрагментов слов. Вдоль горизонтальной оси слова между базовой линий и медианой (рис. 4) выделяется прямоугольная область, и там, где она пересекает символы текста – буквы и некоторые знаки препинания, например, вопросительный/восклицательный знаки – изменяется яркость. Визуально данный эффект похож на перечеркивание слова осветляющим маркером. Обозначим данное преобразование как нанесение *перечеркивающей линии*. При внедрении цифровой метки один бит кодируется одним или несколькими словами для минимизации ошибки. Извлечение метки из документа выполняется построчно, каждое слово в строке проверяется на наличие перечеркивающей линии, для чего применяется обученная нейронная сеть на основе архитектуры U-Net [28]. Стирание метки также выполняется посредством нейронной сети, восстанавливающей оригинальное визуальное представление каждого слова по отдельности.

Для разработанных алгоритмов маркирования была разработана методика тестирования, приближенная к условиям реальной эксплуатации системы. Методика предполагает проверку работы в трех сценариях (*П* – печать, *С* – сканирование, *Ф* – фотографирование): *П-С, П-С-П-С, П-Ф*. На основе открытых источников был сформирован набор изображений документов, содержащих текст различного форматирования, изображения и таблицы. В общей сложности каждый алгоритм маркирования проходит порядка 400 тестов.

Алгоритм №1 демонстрирует наилучшую незаметность метки согласно результатам экспертной оценки. Наилучшую точность извлечения показывает алгоритм №3 (табл. 1). При оценке оценки точности извлечения метки рассчитывались следующие метки:

- *BER (Bit Error Rate)* – отношение числа неверно извлеченных бит цифровой метки к общему числу бит метки для документов, в которых присутствует достаточное для внедрения метки количество машинописного текста (70% документов);
- *Полнота* – доля излеченных меток с $BER = 1$.

Наибольшее значение имеет метрика *полноты*, так как данная метрика позволяет оценить эффективность алгоритма при проведении расследования утечки. Помимо развития каждого из алгоритмов в отдельности, планируется проведение работ по их совместному применению.

Табл. 1. Сравнительная таблица оценки точности алгоритмов маркирования
Table 1. Comparative table of accuracy evaluation of marking algorithms

Тестовый сценарий	*Алгоритм №1*		*Алгоритм №2*		*Алгоритм №3*	
	BER-32	*Полнота*	*BER-32*	*Полнота*	*BER-32*	*Полнота*
П-С	0.8565	0.6622	0.9771	0.8378	0.9991	0.9910
П-С-П-С	0.8110	0.5930	0.9150	0.5964	0.9974	0.9728
П-Ф	0.7297	0.4247	0.8294	0.2857	0.9948	0.9277

4. Анализ атак на систему

В данном разделе рассмотрены основные угрозы и способы противодействия разработанной системе с позиции потенциального внутреннего нарушителя. Атаки на систему можно разделить на следующие категории:

- *Искажение маркированного изображения* – изображение документа перед анонимной публикацией изменяется с целью затруднить или сделать невозможным извлечение ЦВЗ;

- *Подмена метки* – замена встраиваемой в документ метки на отличную от метки пользователя;
- *Обход системы маркирования* – предотвращение внедрения ЦВЗ в документ.

Наиболее обширной областью атак является категория искажения изображения. Стоит отметить, что атаки данного типа могут иметь непреднамеренный характер, поскольку преобразование из цифрового представления (растровое изображение, готовое к печати) в аналоговое (бумажный носитель или экран монитора), а затем снова в цифровое (растровое изображение, полученное сканером или фотоаппаратом) неизбежно ведет к потере части информации. Подобная угроза учитывалась при разработке – тестирование системы показало устойчивость алгоритмов к данной атаке. Тем не менее, искажения могут вноситься в изображение намеренно, причем некоторые искажения обратимы, другие – нет. К обратимым искажениям относятся: искажение перспективы, нарушение контраста и баланса белого. К необратимым искажениям можно отнести искажения, нацеленные на уменьшение общего числа точек растрового изображения (снижение разрешения) или ухудшение цветности (конвертация цветовой схемы). Обратимые искажения аналитик может скорректировать посредством графического редактора, необратимые – ухудшают точность извлечения метки, но в то же время снижают ценность утечки.

Существует возможность утечки текстового содержания документа. Для этого нарушителю потребуется переписать документ вручную или распознать содержимое методами OCR. Трудоемкость организации такой утечки значительно выше, чем утечка изображения. В данном сценарии следует отметить, что ценность утечки может быть ничтожно малой, поскольку в переписанном вручную документе отсутствуют такие элементы, как печать или подпись.

Другая возможная атака – имитация алгоритма стирания с целью полного удаления ЦВЗ или замены метки на метку другого пользователя. Такая атака потребует обратной разработки алгоритма маркирования. Искажение имитации на изображении маркированного документа можно получить при помощи графического редактора или специально созданной программы. Помимо высокой трудоемкости процесса создания имитации, возникает вероятность неполного стирания или замены метки, поскольку на практике крайне сложно полностью скрыть следы редактирования изображения.

Категория атак, связанных с подменой метки на этапе генерации метки в системе, требует доступа к учетной записи и рабочей станции другого пользователя. Как правило, подобные атаки совершаются по неосторожности сотрудников (как пример, стикер с логином и паролем на мониторе) или методами социальной инженерии (рассылка через корпоративную почту). Реализация данной атаки усложняется тем, что требует от злоумышленника подмены не только учетной записи пользователя, но и рабочей станции.

Для реализации атак, подразумевающих обход или отключение системы маркирования, потребуется получение прав суперпользователя. Не секрет, что существует множество уязвимостей повышения прав пользователя. Однако защита от подобных угроз лежит не на разработчиках описываемой системы, а на ИБ специалистах и системных администраторах, ответственных за своевременную установку обновлений безопасности на рабочие станции периметра организации.

По результатам анализа возможных атак на систему была уточнена модель нарушителя. С бóльшей вероятностью нарушитель непреднамеренный – невнимательный или неосведомленный пользователь. Теоретически возможна атака на систему, при которой нельзя установить виновника утечки, однако трудоемкость ее реализации высока, требует от атакующего проведения исследования алгоритмов маркирования, выполняется вручную и плохо поддается автоматизации.

5. Заключение

Разработанная система маркирования документов, выводимых на печать или экран, позволяет противостоять угрозам, с которыми не способны справиться традиционные DLP–системы. Существующие на рынке решения имеют ограничения, затрудняющие внедрение решения в ряд организаций. Представленная система лишена этих ограничений: при извлечении цифровой метки не требуется оригинальный документ, а также предусмотрена функциональность внедрения цифровой метки в маркированные ранее документы. Разработанная система имеет высокую эффективность и показала свою применимость на практике.

Список литературы / References

[1]. Исследование утечек информации ограниченного доступа в 2020 году. InfoWatch. 2021, 40 стр. / Research on restricted information leaks in 2020. InfoWatch. 2021, 40 p. Available at: https://www.infowatch.ru/analytics/analitika/issledovanie-utechek-informatsii-ogranichennogo-dostupa-v-2020-godu, accessed 24.10.2021.

[2]. Утечки информации ограниченного доступа: отчет за 9 месяцев 2020 г. Экспертно-аналитический центр InfoWatch, 2020 г. / Restricted information leaks: report for 9 months of 2020. InfoWatch Analytical Center, 2020 (in Russian). Available at: https://www.infowatch.ru/analytics/analitika/utechki-informatsii-ogranichennogo-dostupa-otchet-za-9-mesyatsev-2020, accessed 24.10.2021.

[3]. McAfee Data Loss Prevention. Available at: https://docs.mcafee.com/bundle/data-loss-prevention-11.4.x-product-guide, accessed 24.10.2021.

[4]. Symantec Data Loss Prevention. Available at: https://www.broadcom.com/products/cyber-security/information-protection/data-loss-prevention, accessed 24.10.2021.

[5]. Trace Doc. Available at: https://secretgroup.ru/trace-doc, accessed 10.08.2021.

[6]. Unique Interface. EveryTag. Available at: https://everytag.ru/ui, accessed 10.08.2021.

[7]. Safe Copy. Available at: https://www.niisokb.ru/products/safecopy, accessed 10.08.2021.

[8]. Козлов С.В., Копылов С.А. и др. Реализация маркирования в подсистеме печати ОС семейства Windows на основе виртуального XPS-принтера. Труды ИСП РАН, том 32, вып. 5, 2020 г., стр. 95-110 / Kozlov S.V., Kopylov S.A. et al. Implementing watermarking based on a virtual XPS printer for Windows operating systems. rudy ISP RAN/Proc. ISP RAS, vol. 32, issue 5, 2020, pp. 95-110 (in Russian). DOI: 10.15514/ISPRAS–2020–32(5)–7.

[9]. Gugelmann D., Sommer D. et al. Screen Watermarking for Data Theft Investigation and Attribution. In Proc. of the 10th International Conference on Cyber Conflict (CyCon), 2018, pp. 391-408.

[10]. Fang H., Zhang W. et al. Screen-Shooting Resilient Watermarking. IEEE Transactions on Information Forensics and Security, vol. 14, no. 6, 2019, pp. 1403-1418.

[11]. Chen W., Ren N. et al. Screen-Cam Robust Image Watermarking with Feature-Based Synchronization. Applied Sciences, vol. 10, no. 21, 2020, article no. 7494.

[12]. Якушев А.Ю., Маркин Ю.В. и др. Маркирование текстовых документов на экране монитора посредством изменения яркости фона в областях межстрочных интервалов. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 147-162 / Yakushev A.Yu., Markin Yu.V. et al. Text documents screen watermarking by changing background brightness in the interline spacing. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021. pp. 147-162 (in Russian). DOI: 10.15514/ISPRAS–2021–33(4)–11

[13]. Pramila A., Keskinarkaus A., Seppänen T. Multiple domain watermarking for print-scan and JPEG resilient data hiding. Lecture Notes in Computer Science, vol. 5041, 2007, pp. 279-293.

[14]. Dong P., Galatsanos N.P. Affine transformation resistant watermarking based on image normalization. In Proc. of the International Conference on Image Processing, 2002, pp. 489-492.

[15]. Ahvanooey M.T., Li Q.et al. Modern text hiding, text steganalysis, and applications: a comparative analysis. Entropy, vol. 21, no. 4, 2019, article no. 355.

[16]. Topkara M., Topkara U., Atallah M.J. Words are not enough: sentence level natural language watermarking. In Proc. of the 4th ACM International Workshop on Contents Protection and Security, 2006, pp. 37-46.

[17]. Topkara U., Topkara M., Atallah M.J. The hiding virtues of ambiguity: quantifiably resilient watermarking of natural language text through synonym substitutions. In Proc. of the 8th Workshop on Multimedia and Security, 2006, pp. 164-174.

[18]. Shirali-Shahreza M. A new Persian/Arabic text steganography using “La” word. In Advances in computer and information sciences and engineering, Springer, 2008, pp. 339-342.

[19]. Low S. H., Maxemchuk N. F. et al. Document marking and identification using both line and word shifting. In Proc. of INFOCOM'95, 1995, pp. 853-860.

[20]. Alattar A. M., Alattar O. M. Watermarking electronic text documents containing justified paragraphs and irregular line spacing. Security, Steganography, and Watermarking of Multimedia Contents VI, vol. 5306, 2004, pp. 685-695.

[21]. Brassil J. T., Low S. Electronic marking and identification techniques to discourage document copying. IEEE Journal on Selected Areas in Communications, vol. 13, no. 8, 1995, pp. 1495-1504.

[22]. Kim Y.W., Moon K.A., Oh I. S. A Text Watermarking Algorithm based on Word Classification and Inter-word Space Statistics. In Proc. of the Seventh International Conference on Document Analysis and Recognition, 2003, pp. 775-779.

[23]. Tan L., Hu K. et al. Print-scan invariant text image watermarking for hardcopy document authentication. Multimedia Tools and Applications, vol. 78, no. 10, 2018, pp. 13189-13211.

[24]. Xiao C., Zhang C., Zheng C. Fontcode: Embedding information in text documents using glyph perturbation. ACM Transactions on Graphics (TOG), vol. 37, no. 2, 2017, pp. 1-16.

[25]. Gutub A., Fattani M. A novel Arabic text steganography method using letter points and extensions. In Proc. of the WASET International Conference on Computer, Information and Systems Science and Engineering (ICCISSE), 2007, pp. 28-31.

[26]. Козачок А.В., Копылов С.А. и др. Алгоритм маркирования текстовых документов на основе изменении интервала между словами, обеспечивающий устойчивость к преобразованию формата. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 131-146 / Kozachok A.V., Kopylov S.A. et al. Text documents marking algorithm based on interword distances shifting invariant to format conversion. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021. pp. 131-146 (in Russian). DOI: 10.15514/ISPRAS–2021–33(4)–10.

[27]. Обыденков Д.О., Фролов А.Е. и др. Методы маркирования текстовых документов при печати посредством вертикального сдвига и изменения яркости фрагментов слов. Труды ИСП РАН, том 33, вып. 5, 2021 г., стр. 65-82 / Obydenkov D. O., Frolov A.E. et al. Printed text documents watermarking based on vertical word shift and word fragments brightness changing. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 5, 2021, pp. 65-82 (in Russian). DOI: 10.15514/ISPRAS–2021–33(5).

[28]. Ronneberger O., Fischer P., Brox T. U-net: Convolutional networks for biomedical image segmentation Lecture Notes in Computer Science, vol. 9351, 2015, pp. 234-241.

Информация об авторах / Information about authors

Дмитрий Олегович ОБЫДЕНКОВ – аспирант. Научные интересы: методы сокрытия и защищенной передачи информации, компьютерные сети, технологии анализа сетевого трафика.

Dmitry Olegovich OBYDENKOV is a graduate student. Scientific interests: methods for information hiding and secure transmission, computer networks, technologies of network traffic analysis.

Алексей Юрьевич ЯКУШЕВ – студент. Научные интересы: стеганография, обработка цифровых изображений, алгоритмы машинного обучения.

Aleksey Yur'evich YAKUSHEV is a student. Scientific interests: steganography, digital image processing, machine learning algorithms.

Юрий Витальевич МАРКИН – научный сотрудник, кандидат технических наук. Область научных интересов: информационная безопасность, анализ сетевого трафика, обработка изображений, алгоритмы машинного обучения.

Yury Vital'evich MARKIN is a researcher, PhD in Technical Sciences. Scientific interests: information security, network traffic analysis, image processing, machine learning algorithms.

Станислав Александрович ФОМИН – ведущий программист. Область научных интересов: теория сложности, алгоритмы дискретной оптимизации, верификация ПО, архитектура информационных систем.

Stanislav Alexandrovich FOMIN – leading programmer. Research interests: complexity theory, discrete optimization algorithms, information systems architecture.

Александр Евгеньевич ФРОЛОВ – студент-магистр. Научные интересы: стеганография, методы анонимизации сетевого трафика, компьютерные сети, методы машинного обучения.

Alexander Evgenevich FROLOV is a master student. Research interests: steganography, traffic anonymization methods, computer networks, machine learning.

Сергей Викторович КОЗЛОВ – кандидат технических наук. Сфера научных интересов: информационная безопасность, защита от несанкционированного доступа, особенности построения и функционирования операционных систем, средства и методы программирования.

Sergey Viktorovich KOZLOV – Candidate of Technical Sciences. Research interests: information security, protection from unauthorized access, construction and functioning features of operating systems, programming tools and methods.

Дмитрий Дмитриевич ГРОМЕЙ – сотрудник Академии Федеральной службы охраны Российской Федерации). Область научных интересов: общая теория систем, системы управления базами данных, машинное обучение, архитектура операционных систем, сетевые технологии.

Dmitry Dmitrievich GROMEY – employer of the Academy of Federal Security Guard Service of the Russian Federation. Research interests: general system theory, database management system, machine learning, operating system architecture, computer network.

Александр Васильевич КОЗАЧОК – доктор технических наук, доцент, сотрудник Академии Федеральной службы охраны Российской Федерации. Его научные интересы включают: информационная безопасность, защита от несанкционированного доступа, математическая криптография, теоретические проблемы информатики.

Alexander Vasilievich KOZACHOK – Doctor of Technical Sciences, Associated Professor. Employer of the Academy of Federal Guard Service. His research interests include: information security, unauthorized access protection, mathematical cryptography and theoretical problems of computer science.

Борис Владимирович КОНДРАТЬЕВ, сфера научных интересов: безопасность информации, защита информации от несанкционированного доступа и утечки по техническим каналам, построение информационных систем в защищенном исполнении, сертификация программного обеспечения по требованиям безопасности информации.

Boris Vladimirovich KONDRAT'EV, scientific interests: information security, protection of information from unauthorized access and leakage through technical channels, building information systems in a secure design, certification of software for information security requirements.

DOI: 10.15514/ISPRAS-2021-33(6)-12

Реализация искусственных нейронных сетей на ПЛИС с помощью открытых инструментов

[1,2] *М.С. Лебедев, ORCID: 0000-0002-0207-7672 <lebedev@ispras.ru>*
[1] *П.Н. Белецкий, ORCID: 0000-0003-2072-958X <belecky@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Российский экономический университет им. Г.В. Плеханова, 117997, Россия, г. Москва, Стремянный пер., д. 36*

Аннотация. Искусственные нейронные сети широко распространены в современном мире. Для их исполнения используются различные устройства: от микропроцессоров до ПЛИС и заказных СБИС. Важной проблемой при этом является ускорение исполнения нейронных сетей. В этой области на данный момент существует множество открытых инструментов. В данной статье содержится обзор нескольких открытых инструментов для исполнения, ускорения нейронных сетей и синтеза аппаратуры по ним. Некоторые из рассмотренных инструментов были выбраны для апробации на ПЛИС. Для этого было разработано пять тестовых моделей нейронных сетей. Процессор Intel, графический процессор NVIDIA и ПЛИС Cyclone V использовались для проведения экспериментов. Результаты показали, что инструменты TVM/VTA и LeFlow оказались способны довести тестовые модели до исполнения на ПЛИС. Однако результаты исполнения показали, что в большинстве случаев ПЛИС проигрывает в быстродействии другим платформам.

Ключевые слова: искусственный интеллект; нейронные сети; специализированные ускорители; высокоуровневый синтез; ПЛИС; открытое программное обеспечение.

Для цитирования: Лебедев М.С., Белецкий П.Н. Реализация искусственных нейронных сетей на ПЛИС с помощью открытых инструментов. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 175-192. DOI: 10.15514/ISPRAS–2021–33(6)–12

Artificial Neural Network Inference on FPGAs Using Open-Source Tools

[1,2] *M.S. Lebedev, ORCID: 0000-0002-0207-7672 <lebedev@ispras.ru>*
[1] *P.N. Belecky, ORCID: 0000-0003-2072-958X <belecky@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*
[2] *Plekhanov Russian University of Economics, 36, Stremyanny lane, Moscow, 117997, Russia.*

Abstract. Artificial neural networks are widely spread in the modern world. Various hardware is used for neural network inference: from CPUs and GPUs to FPGAs and ASICs. An important research area is inference acceleration. Many open-source tools have been proposed in this area. This article contains a review of a range of open-source tools for neural network inference, acceleration and hardware synthesis. Some of the tools have been selected for evaluation on an FPGA. Five neural network examples have been used as test models. Intel CPU, NVIDIA GPU and Cyclone V FPGA have been used as evaluation platforms. Results show that TVM/VTA and LeFlow tools can successfully process neural network models and run them on the FPGA. However, execution results are controversial.

Keywords: artificial intelligence; neural networks; custom accelerators; high-level synthesis; FPGA; open-source

For citation: Lebedev M.S., Belecky P.N. Artificial Neural Network Inference on FPGAs Using Open-Source Tools. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 175-192 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–12

1. Введение

Искусственный интеллект все больше проникает в повседневную жизнь. Нейронные сети широко используются в обработке мультимедиа, медицине, беспилотном транспорте, общественной безопасности и т.д. В настоящее время уже существует множество открытых (*open-source*) инструментов и библиотек для разработки, хранения, распространения и исполнения нейронных сетей. Наиболее значимые из них – TensorFlow [1], PyTorch [2] и Caffe [3]. А формат ONNX [4] часто применяется для хранения и передачи моделей нейронных сетей и поддерживается большим количеством инструментов.

Важными проблемами в сфере искусственных нейронных сетей являются их оптимизация, исполнение и ускорение на различных аппаратных платформах: мобильных и стационарных микропроцессорах, графических процессорах, тензорных процессорах, ускорителях машинного обучения, а также ПЛИС и проблемно-ориентированных СБИС.

Решением проблемы ускорения может быть использование особых видов нейронных сетей: разреженных [5][6] или бинаризованных [7][8]. Также возможна разработка новых алгоритмов вычислений [9][10] (например, на основе свертки Винограда), техник удаления [11] или слияния [12] слоев нейронной сети, квантования весов [13][14], новых архитектур [15][16] ускорителей и методов оптимизации [17][18] моделей нейронных сетей. Разработчики многих предлагаемых методов заявляют о существенном ускорении вычислений нейронных сетей, в том числе на ПЛИС.

Существует множество открытых инструментов для оптимизации, исполнения и ускорения нейронных сетей, некоторые из которых реализуют описанные выше подходы. Эти инструменты могут быть условно поделены на три группы: 1) инструменты, оптимизирующие модели нейронных сетей для исполнения на устройстве с фиксированной архитектурой; 2) инструменты, оптимизирующие модели нейронных сетей для исполнения на специализированном сопроцессоре (расположенном на ПЛИС); 3) инструменты, позволяющие синтезировать RTL-модель исходной нейронной сети для последующего исполнения на ПЛИС или реализации в виде СБИС.

В этой статье приведен обзор наиболее популярных на сегодняшний день открытых инструментов исполнения и ускорения нейронных сетей, а также результаты экспериментальной апробации нескольких из них на примере моделей двух обученных нейронных сетей (полносвязной и сверточной), а также трех синтетических примеров матричного умножения. Заметим, что процесс обучения нейронных сетей в данной работе не рассматривается.

В разд. 2 данной статьи приведен короткий обзор современных форматов и библиотек для представления моделей нейронных сетей. Разд. 3 посвящен обзору инструментов исполнения и ускорения нейронных сетей. В разд. 4 описаны тестовые модели, с помощью которых проводилась апробация выбранных инструментов. В разд. 5 приведены результаты экспериментов. Разд. 6 – заключение.

2. Форматы представления нейронных сетей

В табл. 1 приведены рассматриваемые в статье форматы и библиотеки представления моделей нейронных сетей.

2.1 ONNX

ONNX (Open Neural Network Exchange) [4] – де-факто общепринятый формат хранения и передачи моделей нейронных сетей. Разрабатывается сообществом ведущих технологических компаний, таких как Intel, AMD, Qualcomm, ARM, Google и Microsoft. Большинство представленных в этой статье инструментов поддерживает входные модели в этом формате. ONNX представляет нейронную сеть в виде графа вычислений, состоящего из входов и выходов, операционных слоев, переменных, типов данных и т.д. Модель нейронной сети сохраняется в двоичном виде. Значения весов и смещений закодированы непосредственно в графе вычислений.

Табл. 1. Форматы представления нейронных сетей
Table 1. Neural Network Frameworks and Formats

Название	Лицензия	Разработчик	Год начала разработки	Формат
ONNX	Apache 2.0	Сообщество компаний	2017	Двоичный
TensorFlow	Apache 2.0	Google	2015	Текстовый / двоичный
Keras	Apache 2.0	Google	2015	Текстовый / двоичный
PyTorch	BSD	Facebook	2016	Текстовый / двоичный
Caffe	BSDv2	Калифорнийский университет в Беркли	2014	Текстовый / двоичный
Caffe2	BSD	Facebook	2017	Текстовый / двоичный
CNTK	MIT	Microsoft	2015	Текстовый / двоичный
CoreML	BSDv3	Apple	2017	Двоичный
MXNet	Apache 2.0	Apache	2016	Текстовый / двоичный
Theano	BSDv3	Монреальский университет	2011	Текстовый / двоичный

2.2 TensorFlow/Keras

TensorFlow [1] – созданная компанией Google платформа разработки приложений машинного обучения. TensorFlow позволяет пользователям описывать, обучать, сохранять и исполнять модели нейронных сетей, а также выполнять тензорные вычисления. Платформа состоит из нескольких библиотек и инструментов, покрывающих различные аспекты разработки приложений машинного обучения. Большинство рассмотренных нами инструментов также поддерживает входные модели в формате TensorFlow. В TensorFlow существует формат хранения моделей SavedModel, позволяющий сохранить полную структуру нейронной сети, включая веса и подграфы.

Модели на TensorFlow описываются по слоям и могут быть оптимизированы и исполнены либо с помощью встроенной среды запуска, либо с помощью компилятора XLA (см. раздел 3). Для исполнения моделей TensorFlow использует *«активное исполнение»* (*eager execution*), т.е. операции в нейронной сети выполняются немедленно, без построения графа вычислений.

Keras [19] – часть платформы TensorFlow, ориентированная на быструю разработку моделей нейронных сетей, их исполнение и портирование на другие платформы. В Keras используется формат H5 для хранения моделей нейронных сетей.

2.3 PyTorch

PyTorch [2] – платформа для тензорных вычислений и разработки моделей нейронных сетей, разработанная компанией Facebook. Приложения, разработанные на PyTorch, в основном ориентированы на исполнение на графических процессорах. Модели нейронных сетей в PyTorch основаны на *обратном автодифференцировании* [20] (*reverse-mode auto-differentiation*), позволяющем динамически менять поведение этих сетей. Модели на PyTorch могут быть сохранены в формате Python *pickle* или формате ONNX.

2.4 Caffe/Caffe2

Caffe [3] – платформа для машинного обучения, разработанная в Калифорнийском университете в Беркли (UC Berkeley). Структура нейронной сети в Caffe описывается слой за слоем в текстовом виде, а не в виде программы. Каждый слой описывает отдельную операцию (от непосредственно слоя нейронной сети или функции активации до входов данных и вспомогательных функций). Данные между слоями передаются с помощью специального представления массивов. Если данные передаются от входа к выходу сети, то осуществляется вычисление результата. Если данные передаются в обратном направлении, то выполняется обучение. Обученные нейронные сети сохраняются в двоичном формате.

Caffe2 [21] – дальнейшее развитие платформы Caffe. Разрабатывается компанией Facebook и входит в состав платформы PyTorch.

2.5 CNTK

CNTK (Microsoft Cognitive Toolkit) [22] – набор инструментов машинного обучения, разрабатываемый компанией Microsoft. Модели нейронных сетей в этом наборе представляются в виде направленных графов вычислений. Листовые вершины этих графов представляют собой входные значения и параметры, внутренние вершины – различные матричные операции.

2.6 CoreML

CoreML [23] – формат представления нейронных сетей в программах и на устройствах компании Apple. С этим форматом предоставляется набор инструментов, с помощью которого можно преобразовывать модели нейронных сетей из сторонних форматов в формат CoreML, а также оптимизировать и запускать их на устройствах Apple.

2.7 MXNet

MXNet [24] – платформа машинного обучения, разрабатываемая сообществом программистов и поддерживаемая фондом Apache. Одной из главных особенностей этой платформы является возможность символического программирования и исполнения нейронных сетей для их оптимизации. Модель на MXNet может быть сериализована в виде двух файлов: структуры нейронной сети и ее весов.

2.8 Theano

Theano [25] – библиотека языка Python для тензорных и матричных вычислений, разработанная в Монреальском университете (Université de Montréal). Ее разработка была прекращена, а результаты использованы в новом проекте по оптимизации тензорных вычислений Aesara [26].

3. Инструменты исполнения и ускорения нейронных сетей

В табл. 2 представлены рассматриваемые в данной статье открытые инструменты исполнения и ускорения нейронных сетей. Их можно условно разделить на три группы.

1) Инструменты, оптимизирующие и исполняющие модели нейронных сетей на целевых устройствах с фиксированными архитектурой и набором инструкций, таких как микропроцессоры, графические процессоры, тензорные процессоры, процессоры цифровой обработки сигналов, микроконтроллеры и т.д. В качестве результата своей работы эти инструменты выдают исполняемый файл под целевую архитектуру, либо некоторое промежуточное представление (например, граф вычислений, веса и представление операций), которое может быть исполнено с помощью среды времени выполнения на целевом устройстве. Инструменты, представленные в этой категории: OpenVINO, PlaidML, MACE, XLA, Glow, TVM.

2) Инструменты, оптимизирующие и исполняющие модели нейронных сетей на специальных сопроцессорах-ускорителях, размещенных на ПЛИС. Обычно эти инструменты предоставляют среду времени выполнения, управляющую передачей входных данных на сопроцессор и считыванием результатов вычислений. В данной категории представлены: TVM с сопроцессором VTA и Vitis AI с семейством ядер DPU.

3) Инструменты, позволяющие синтезировать непосредственно RTL-модель нейронной сети, либо транслирующие ее в формат, пригодный для генерации RTL-модели (например, C/C++ или LLVM) с помощью инструментов высокоуровневого синтеза. Эта категория состоит из: LeFlow (трансляция в LLVM и синтез с помощью открытого инструмента LegUp 4.0 [27]), hls4ml, FINN, ONNC (трансляция в C++ и синтез с помощью коммерческого инструмента Vivado [28]), NNgen (непосредственно синтез Verilog).

Табл. 2. Инструменты исполнения и ускорения нейронных сетей
Table 2. Neural Network Acceleration Tools

Инструмент	**Лицензия**	**Разработчик**	**Год начала разработки**
OpenVINO	Apache 2.0	Intel	2018
PlaidML	Apache 2.0	Intel	2017
MACE	Apache 2.0	Xiaomi	2017
TVM	Apache 2.0	Вашингтонский университет, Apache	2017
XLA	Apache 2.0	Google	2017
LeFlow	BSD	Университет Британской Колумбии	2018
Glow	Apache 2.0	Facebook	2017
hls4ml	Apache 2.0	ЦЕРН и др.	2017
NNgen	Apache 2.0	Шинья Такамаэда-Ямадзаки	2017
ONNC	BSDv3	Skymizer	2018
Vitis AI	Apache 2.0	Xilinx	2020
Finn	BSDv3	Xilinx	2018

3.1 OpenVINO

OpenVINO [29] – набор инструментов для ускорения нейронных сетей, разрабатываемый компанией Intel. В качестве целевых устройств поддерживаются только устройства Intel: процессоры, графические процессоры и процессоры компьютерного зрения (Vision

Processing Units, VPUs). Некоторые версии OpenVINO также поддерживали ПЛИС Intel Arria. На вход OpenVINO может принимать модели нейронных сетей в форматах ONNX, TensorFlow, Caffe, MXNet и Kaldi [30]. Входные модели транслируются в промежуточное представление и оптимизируются: удаляются лишние слои и группируются операции. Далее модель нейронной сети компилируется в исполняемый файл для целевой архитектуры.

3.2 PlaidML

PlaidML [31] – тензорный компилятор, разрабатываемый компанией Intel. Способен компилировать модели нейронных сетей в форматах ONNX, Keras и nGraph [32] для запуска на графических процессорах Intel, AMD и NVIDIA. Для процессоров NVIDIA позволяет не строить промежуточное представление модели на CUDA.

3.3 MACE

MACE (Mobile AI Compute Engine) [33] – платформа для исполнения и ускорения нейронных сетей, разрабатываемая компанией Xiaomi и в основном ориентированная на мобильные устройства (процессоры ARM, графические процессоры Adreno и т.д.). На вход MACE может принимать модели нейронных сетей в форматах ONNX, TensorFlow и Caffe. Встроенный интерпретатор обрабатывает граф вычислений и тензорные операции и передает их среде времени выполнения. Эта среда распределяет вычисления между целевыми устройствами.

3.4 TVM

TVM [34] – компилятор нейронных сетей, разработанный Вашингтонским университетом (University of Washington) и ныне поддерживаемый фондом Apache. TVM позволяет компилировать модели нейронных сетей под различные устройства, такие как микропроцессоры, графические процессоры, микроконтроллеры и т.д. С помощью сопроцессора VTA (Versatile Tensor Accelerator) [35] можно осуществлять исполнение нейронных сетей на ПЛИС. TVM предоставляет средства адаптации компилятора под произвольную целевую архитектуру.

TVM может принимать на вход наибольшее число форматов моделей нейронных сетей из всех рассмотренных в этой статье инструментов: ONNX, TensorFlow, Keras, PyTorch, MXNet, CoreML и другие. Входные модели транслируются во внутреннее представление TVM Relay IR, состоящее из множества *функций* – разновидностей графов вычислений с потоком управления, рекурсией и поддержкой сложных структур данных. После оптимизации функции делятся на *подфункции*, или *сегменты*, состоящие из низкоуровневых операций. Структура исходной модели сохраняется в виде последовательности вызовов подфункций. Каждая подфункция может быть оптимизирована и скомпилирована независимо от других. Наиболее низкоуровневые и архитектурно-зависимые оптимизации проводятся уже компиляторами под целевую архитектуру (например, LLVM или CUDA C). Затем сегменты транслируются в программный код (на C, CUDA, OpenCL, LLVM). Последовательность вызовов преобразуется в формат JSON, а веса сохраняются в двоичном формате. Модель целиком может быть сохранена в виде разделяемой библиотеки и загружена средой времени выполнения TVM, работающей на целевом устройстве и управляющей исполнением модели.

3.4.1 VTA

Сопроцессор VTA – это небольшое тензорное ядро, состоящее из следующих элементов (см. рис. 1):

1) модуль выборки инструкций (Instruction Fetch Module), загружающий инструкции сопроцессора из памяти (DRAM), декодирующий их и распределяющий по очередям;
2) модуль загрузки (Load Module), загружающий входные данные и веса из памяти;

3) модуль сохранения (Store Module), загружающий в память результаты вычислений;

4) вычислительный модуль (Compute Module), осуществляющий перемножение матриц и тензорные операции и, в свою очередь, состоящий из:

   a) регистрового файла (Register File);

   b) кэша микроопераций (Micro-op Cache);

   c) тензорного АЛУ (Tensor ALU);

   d) блока умножения матриц (GEMM Core);

   e) очередей и буферов.

Набор инструкций сопроцессора VTA состоит из операций загрузки и сохранения, умножения матриц и арифметических операций над тензорами.

Внутренние параметры сопроцессора могут быть сконфигурированы: размер элемента входных данных, размер элемента матрицы весов, размеры буферов и т.д.

Существуют две реализации сопроцессора VTA для различных отладочных плат. Реализация на C++ ориентирована на синтез с помощью коммерческого инструмента высокоуровневого синтеза Vivado для плат Xilinx PYNQ (система-на-кристалле Zynq-7000) и Avnet Ultra96 (система-на-кристалле Xilinx Zynq UltraScale+). Реализация на языке Chisel ориентирована на синтез с помощью инструмента Quartus 18.1 для платы Terasic DE10-Nano (ПЛИС Intel Cyclone V).

Для работы с сопроцессором VTA требуется среда времени выполнения, работающая на центральном процессоре платы и осуществляющая получение, обработку и передачу в память DRAM входных данных и получение результатов вычислений из нее.

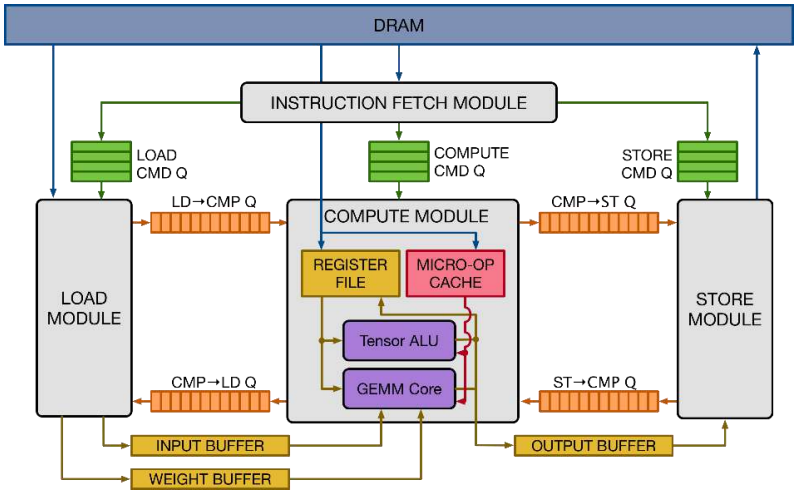


Рис. 1. Структура сопроцессора VTA
Fig. 1. VTA core structure

3.5 XLA

XLA [36] – оптимизирующий компилятор нейронных сетей, входящий в состав платформы TensorFlow. XLA позволяет ускорять исполнение нейронных сетей по сравнению со стандартной средой времени выполнения TensorFlow. Среда времени выполнения TensorFlow представляет операции нейронной сети в виде вычислительных ядер без построения графа вычислений. XLA, напротив, анализирует и оптимизирует граф

вычислений нейронной сети. Он может сливать несколько операций в одно вычислительное ядро и позволяет уменьшить количество обращений в память путем хранения данных в регистрах целевого устройства. XLA осуществляет статическую (ahead-of-time) компиляцию, что позволяет избежать использования среды времени выполнения.

XLA может компилировать модели нейронных сетей в форматах TensorFlow, PyTorch, Julia [37], JAX [38], Nx [39]. Входная модель транслируется во внутреннее представление XLA HLO (High Level Operations), которое затем оптимизируется. Далее HLO-модель трансформируется в LLVM-модель для целевой архитектуры. В качестве целевых архитектур выступают графические процессоры NVIDIA и различные архитектуры микропроцессоров.

3.6 LeFlow

LeFlow [40] – инструмент и маршрут высокоуровневого синтеза RTL-моделей нейронных сетей, разработанный в Университете Британской Колумбии (University of British Columbia). Маршрут (см. рис. 2) позволяет синтезировать Verilog-модель по TensorFlow-модели нейронной сети. Сначала TensorFlow-модель компилируется в неоптимизированное LLVM-представление с помощью XLA. Затем инструмент LeFlow трансформирует и оптимизирует это представление. Наконец, открытый инструмент высокоуровневого синтеза LegUp 4.0 синтезирует RTL-модель нейронной сети. Синтез с помощью LegUp представлен на рис. 2 синими прямоугольниками.

LeFlow трансформирует высокоуровневое представление модели нейронной сети в более приближенное к аппаратуре представление. Добавляются структуры данных для представления тактового сигнала, сигналов сброса, начала работы, завершения работы, выходных данных, создаются глобальные регистры для оптимизации обращений в память и т.д. Так как инструмент LegUp 4.0 не поддерживает некоторые LLVM-операции, сгенерированные XLA, то LeFlow поддерживает несколько измененную версию TensorFlow 1.6 для избегания этих операций. Разработчики LeFlow также добавили несколько собственных оптимизаций в инструмент LegUp 4.0.

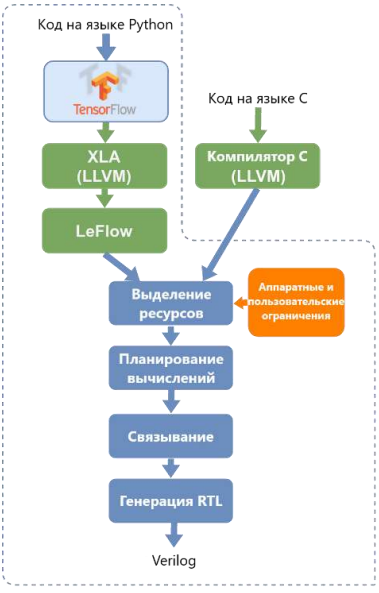


Рис. 2. Маршрут синтеза LeFlow
Fig. 2. LeFlow synthesis route

3.7 Glow

Glow [41] – компилятор нейронных сетей, часть платформы PyTorch. Поддерживает компиляцию под микропроцессоры и графические процессоры. Glow может принимать на вход модели нейронных сетей в форматах ONNX и Caffe2. На основе входной модели строится двухуровневое внутреннее представление. Первое, высокоуровневое, представление – это граф вычислений, сходный с Caffe2-представлением. Операции нейронной сети представляются в виде одной или нескольких вершин графа вычислений. Затем Glow проводит упрощение вершин: операционные вершины трансформируются в вершины низкоуровневых операций линейной алгебры. Далее строится второе, низкоуровневое, представление. На этом уровне операции трансформируются в инструкции. И, наконец, Glow генерирует машинный код. На каждом уровне внутреннего представления Glow проводит различные оптимизации графа вычислений.

3.8 hls4ml

hls4ml [42] – инструмент синтеза C++-модели нейронной сети для инструмента Vivado, разрабатываемый сообществом исследователей из ЦЕРН и нескольких университетов США. hls4ml может обработать модели нейронных сетей в форматах ONNX, Keras, TensorFlow и PyTorch. Инструмент использует библиотеку RFNoC [43] для представления операций нейронной сети. Эта библиотека содержит в себе набор шаблонов нейросетевых операций на C++, оптимизированных для синтеза с помощью прагм инструмента Vivado. hls4ml строит граф вычислений на основе входной модели и вычисляет параметры инстанциации операций. Помимо выходной C++-модели, инструмент генерирует управляющие скрипты для Vivado.

3.9 NNgen

NNgen [44] – инструмент синтеза RTL-моделей из высокоуровневых моделей нейронных сетей, разработанный Шинья Такамаэда-Ямадзаки из Токийского университета (University of Tokyo). NNgen принимает на вход модели в формате ONNX, либо в виде описания на основе собственного предметно-ориентированного языка и библиотеки языка Python.

3.10 ONNC

ONNC [45] – компилятор нейронных сетей под разные целевые устройства (микро- и графические процессоры, процессоры цифровой обработки данных, СБИС и т.д.), разработанный компанией Skymizer. В качестве входного формата нейронных сетей ONNC поддерживает формат ONNX. Инструмент транслирует входную модель в промежуточное представление, оптимизирует ее и разделяет на части, планирует вычисления и выделяет необходимую память. Затем промежуточное представление транслируется в битовый код LLVM для целевой архитектуры. ONNC может транслировать модель нейронной сети в код на языке C, используя собственные реализации операций.

3.11 Vitis AI

Vitis AI [46] – среда для исполнения нейронных сетей на ПЛИС Xilinx, разработанная этой же компанией. Vitis AI состоит из компилятора, оптимизатора, квантователя, профилировщика, среды времени выполнения и набора сопроцессоров глубокого обучения DPU (Deep Learning Processing Unit). Vitis AI не является полностью открытым: для оптимизатора требуется коммерческая лицензия, а сопроцессоры DPU зашифрованы и практически все доступны только в платной версии инструмента Vivado. Целевыми платформами для Vitis AI являются системы-на-кристалле Alveo, Zynq UltraScale+ и ограниченно Zynq-7000. На вход Vitis AI может принимать модели нейронных сетей в форматах TensorFlow, Caffe и PyTorch. Это единственный инструмент из рассмотренных, не

поддерживающий формат ONNX. Для общения с сопроцессором DPU Vitis AI использует среду времени выполнения XRT [47].

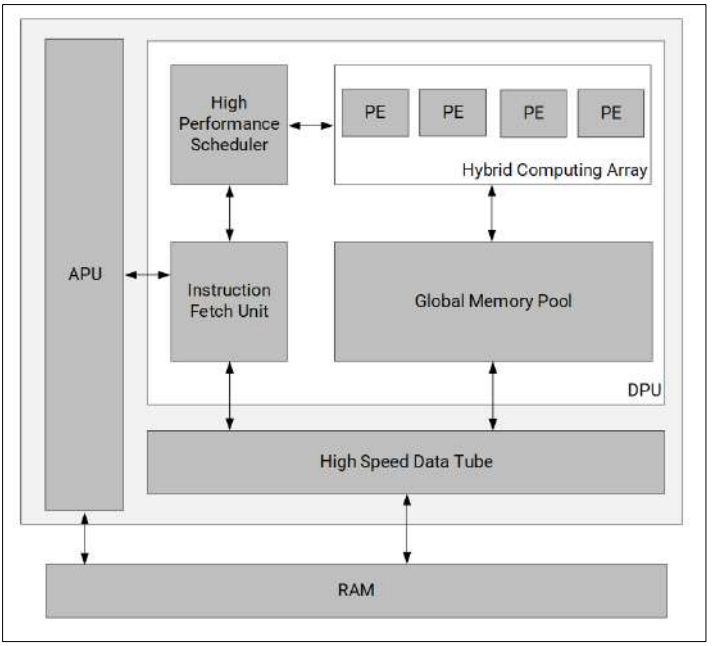


Рис. 3. Структура сопроцессора DPU
Fig. 3. DPU core structure

3.11.1 Deep Learning Processor Unit

Простейшее сопроцессорное ядро DPU для ПЛИС Zynq-7000 или Zynq UltraScale+ представлено на рис. 3 и состоит из следующих компонентов:

1) пула глобальной памяти (Global Memory Pool);
2) устройства выборки инструкций (Instruction Fetch Unit);
3) планировщика вычислений (High Performance Scheduler);
4) гибридного массива вычислителей (Hybrid Computing Array of Processing Engines);
5) внешних компонентов:
   a) блока управления приложением (Application Processing Unit, APU), контролирующего прерывания и передачу данных;
   b) высокоскоростного канала передачи данных (High Speed Data Tube).

Ядро DPU может быть сконфигурировано под определенную нейронную сеть с помощью выбора ширины входных и выходных каналов, функции активации и т.д. Для более продвинутых ПЛИС Xilinx существует ряд более мощных ядер DPU, ориентированных, например, на быструю обработку изображений.

3.12 FINN

FINN [48] – инструмент высокоуровневого синтеза моделей нейронных сетей, разработанный компанией Xilinx для ПЛИС этой же компании. Инструмент ориентирован на квантованные нейронные сети и принимает на вход модели в форматах PyTorch и ONNX. FINN оптимизирует операции с плавающей точкой и разделяет модель на синтезируемые и несинтезируемые слои. Синтезируемые слои реализуются с помощью библиотеки finn-hlslib [49]. Затем они оптимизируются, разделяются на сложнофункциональные блоки и

синтезируются в двоичный образ для ПЛИС с помощью инструмента Vivado. FINN также предоставляет среду времени выполнения, контролирующую процесс вычислений на ПЛИС.

4. Тестовые модели

4.1 MNIST-FC

MNIST-FC – полносвязная нейронная сеть, состоящая из одного слоя и распознающая рукописные цифры на изображениях набора MNIST. Имеет 784 входа (картинка 28×28 пикселей) и 10 выходов (каждый выход соответствует цифре от 0 до 9). Функция активации выходного слоя – *softmax*. Модель MNIST-FC была реализована и обучена на Keras и TensorFlow 1.6 (для инструмента LeFlow).

4.2 MNIST-CNN

MNIST-CNN – сверточная нейронная сеть, распознающая рукописные цифры из набора MNIST. Имеет такое же количество входов и выходов, как и модель MNIST-FC. Модель MNIST-CNN состоит из следующих слоев:

1) сверточный слой с функцией активации ReLu;
2) слой подвыборки;
3) полносвязный слой с функцией активации ReLu;
4) слой дропаута;
5) выходной слой без функции активации.

Модель MNIST-CNN была реализована и обучена на Keras и TensorFlow 1.6.

4.3 Синтетические примеры

Следующие три модели осуществляют умножение матриц и являются нагрузочными примерами для инструмента LeFlow (не осуществляют осмысленных вычислений). Они были реализованы только на TensorFlow 1.6.

1) MULT-0 – представляет собой умножение вектора размером N на матрицу размером N×N (по умолчанию, N=100)

2) MULT-10-R – синтетическая нейронная сеть из 10 полносвязных слоев. Каждый слой имеет 100 входов и 100 выходов и функцию активации *ReLu*. Обучение сети не производилось.

3) MULT-10-S – идентична MULT-10-R за исключением использования *сигмоида* в качестве функции активации последнего слоя.

5. Экспериментальная апробация

5.1 Эксперименты

Эксперименты проводились с помощью трех отладочных плат: Terasic DE10-Standard [50] и Terasic DE1-SoC [51], обе на базе ПЛИС Cyclone V, и платы Zybo Z7-20 [52] (система-на-кристалле Zynq-7000). Исходные модели нейронных сетей на Keras и TensorFlow исполнялись на ПК с процессором Intel Core i7-6700 3,4 ГГц (далее – CPU), 32 Гб RAM и ОС Ubuntu 20.04. Нейронные сети, оптимизированные с помощью инструмента TVM, исполнялись также на графическом процессоре NVIDIA GeForce GTX 770 (далее – GPU).

Главной целью экспериментов была реализация (и по возможности ускорение) моделей нейронных сетей на ПЛИС. Для этого были выбраны следующие инструменты: hls4ml, ONNC, Vitis AI, LeFlow и TVM (с сопроцессором VTA). Инструмент FINN не рассматривался из-за тесной интеграции с Vivado. Инструмент OpenVINO не участвовал в экспериментах из-

за ориентации на ПЛИС Intel Arria. Инструмент NNgen выдавал ошибки при обработке ONNX-моделей и был исключен из рассмотрения. До исполнения нейронной сети на ПЛИС удалось довести только два инструмента: LeFlow и TVM.

В процессе экспериментов замерялось непосредственно время исполнения нейронной сети, без времен обучения, компиляции и оптимизации. Также были получены показатели потребления ресурсов ПЛИС: количество использованных адаптивных логических модулей (ALM), процессоров обработки сигналов (DSP), регистров и блоков памяти, и максимальная частота синтезированных схем.

5.1.1 hls4ml

Инструмент успешно синтезировал C++-модели нейронных сетей MNIST-FC и MNIST-CNN, сохраненных в формате Keras H5. Затем была проведена безуспешная попытка синтезировать двоичные образы для ПЛИС с помощью инструмента Vivado 2020.1. Несмотря на то, что тестовые модели нейронных сетей являются простыми, оказалось, что количество связей между нейронами (представленными в виде циклов for в C++-модели) слишком велико для инструмента Vivado. При обработке модели MNIST-FC инструмент выдал сообщение, что не способен обработать подобный цикл for со слишком большим числом итераций. При обработке MNIST-CNN инструмент и вовсе завис на этапе оптимизации кода. Анализ кода MNIST-CNN показал, что в нем содержатся циклы с еще бóльшим количеством итерация, чем в коде MNIST-FC.

5.1.2 ONNC

Инструмент успешно синтезировал C++-модели MNIST-FC и MNIST-CNN, переданные ему в формате ONNX. Однако оказалось, что реализации слоев нейронных сетей скрыты во внутренней библиотеке ONNC, поэтому синтез двоичного образа для ПЛИС оказался невозможен.

Табл. 3. Параметры синтезированных моделей
Table 3. Synthesized Model Parameters

Параметр	**Модель**			
	MNIST-FC	***MULT-0***	***MULT-10-R***	***MULT-10-S***
Плата	DE1-SoC		DE10-Standard	
Размер RTL-модели, строк кода	5 240	1 252	6 011	11 490
Использование ALM, штук	5 331 (17%)	917 (3%)	4 598 (11%)	11 356 (27%)
Использование DSP, штук	1 (1%)	4 (5%)	31 (28%)	32 (29%)
Использование регистров, штук	7 311	1 554	6 842	16 277
Использование памяти, битов	278 986 (7%)	326 820 (8%)	3 213 220 (57%)	3 216 138 (57%)
Максимальная частота, МГц	127.39	122.55	96.08	93.63

5.1.3 Vitis AI

Эксперименты с Vitis AI не были проведены из-за отсутствия моделей сопроцессоров DPU в бесплатной версии Vivado. В качестве примера Xilinx предоставляет зашифрованный IP-блок на основе нескольких простых сопроцессоров DPU, однако он оказался слишком велик для имевшейся у нас в распоряжении отладочной платы Zybo Z7-20.

5.1.4 LeFlow

Инструмент синтезировал RTL-модели нейронной сети MNIST-FC и синтетических нейронных сетей, но не справился с моделью сверточной нейронной сети MNIST-CNN. Двоичные образы для ПЛИС были синтезированы с помощью инструмента Quartus [53] версии 20.1. В табл. 3 представлены параметры полученных RTL-моделей и соответствующих двоичных образов. Плата DE10-Standard была использована в случаях, когда синтезированная модель не помещалась на плате DE1-SoC.

Синтезированные RTL-модели были исполнены на симуляторе с целью получения их логического времени исполнения в тактах. Эти данные и значения максимальных частот были использованы для подсчета минимального времени исполнения нейронных сетей на ПЛИС. Также было измерено время исполнения исходных TensorFlow-моделей на CPU. Производительность полученных моделей представлена в табл. 4.

Табл. 4. Производительность синтезированных моделей
Table 4. Synthesized Model Performance

Параметр	**Модель**			
	MNIST-FC	***MULT-0***	***MULT-10-R***	***MULT-10-S***
Плата	DE1-SoC		DE10-Standard	
Логическое время исполнения, тактов	223 590	280 303	2 808 023	2 812 513
Время исполнения на CPU, мс	4	1	5	5
Время исполнения на ПЛИС, мс	1.76	2.29	29.23	30.03
Коэффициент ускорения (на ПЛИС)	2.28	0.44	0.17	0.17

5.1.5 TVM

Ранее было упомянуто, что сопроцессор VTA имеет версию для платы Terasic DE10-Nano, которая отличается от DE10-Standard моделью ПЛИС Cyclone V и некоторым периферийным оборудованием. Поэтому мы адаптировали ядро VTA для DE10-Standard путем изменения сборочных скриптов.

Табл. 5. Параметры двоичного образа сопроцессора VTA
Table 5. VTA Core Bitstream Parameters

Параметр	**Значение**
Плата	DE10-Standard
Использование ALM, штук	20 311 (48%)
Использование DSP, штук	0 (0%)
Использование регистров, штук	19 226
Использование памяти, битов	3 905 016 (69%)
Максимальная частота, МГц	137.85

Более сложной задачей было установить среду времени выполнения TVM на нашу плату, так как стандартные версии ОС Linux не подходили для сборки среды. Для этого был собран образ ОС Linux на основе ядра версии 4.14.130 [54] и файловой системы Ubuntu Base [55]

версии 18.04.5, а также найден подходящий драйвер [56] резервирования непрерывной области памяти (Contiguous Memory Allocator, CMA).

Двоичный образ сопроцессора VTA был синтезирован с помощью инструмента Quartus 18.1. Параметры образа представлены в табл. 5.

Инструмент TVM скомпилировал, оптимизировал и исполнил на ПЛИС обе модели нейронных сетей MNIST-FC и MNIST-CNN. Примеры, разработанные на TensorFlow 1.6, не были обработаны TVM, так как он не поддерживает столь старую версию TensorFlow. Во время экспериментов измерялись времена исполнения оптимизированных моделей на CPU, GPU и VTA. Время исполнения исходных Keras-моделей было также измерено. Модель MNIST-CNN была квантована и исполнена с помощью средств TVM. Производительность полученных моделей представлена в табл. 6. Ускорение вычислялось в сравнении с исходной Keras-моделью.

Табл. 6. Производительность сопроцессора VTA
Table 6. VTA Core Performance

Параметр		**Модель**		
		MNIST-FC	***MNIST-CNN***	***MNIST-CNN (с квантованием)***
Время исполнения на CPU, мс	Keras	0.38	2.55	н/д
	TVM	0.0037	0.8	0.44
Время исполнения на GPU, мс	TVM	0.02	0.17	Ошибка CUDA
Время исполнения на VTA, мс		0.10	105.0	33.3
Коэффициент ускорения (на сопроцессоре)	Keras	3.8	0.024	0.08
	TVM, CPU	0.037	0.008	0.014
	TVM, GPU	0.020	0.0016	н/д

5.2 Анализ

Эксперименты показали, что большинство открытых инструментов способны обрабатывать модели нейронных сетей. Однако некоторые из них не могут быть использованы для исполнения нейронных сетей на ПЛИС (ONNC, hls4ml). Некоторые инструменты зависят от коммерческих продуктов (hls4ml, Vitis AI и др.). Только TVM и LeFlow оказались способны исполнить (или синтезировать) тестовые модели на ПЛИС. Но LeFlow не синтезировал сверточную нейронную сеть MNIST-CNN.

Исполнение моделей нейронных сетей на ПЛИС показало неоднозначные результаты. Простая полносвязная сеть MNIST-FC была ускорена по сравнению с исходной Keras/TensorFlow-моделью. С другой стороны, другие модели исполнялись существенно медленнее даже по сравнению с исходными моделями. Оптимизированные с помощью TVM модели показали лучшее время исполнения на CPU (для MNIST-FC) и GPU (для MNIST-CNN). Модель MNIST-CNN, исполненная на VTA, оказалась в сотни раз медленнее, чем исполненная на CPU или GPU. Квантование и упаковка улучшили производительность модели на VTA, но несущественно. Что касается LeFlow, производительность синтезированных схем почти полностью зависит от инструмента LegUp.

Слабые результаты, показанные сопроцессором VTA для сверточной нейронной сети, можно объяснить неудачно подобранными стандартными настройками ядра, плохо совместимыми с моделью нейронной сети. Эта тема требует дальнейшего исследования. Также к недостаткам сопроцессора VTA можно отнести его небольшие размеры и, как следствие, необходимость частой пересылки больших объемов данных между микропроцессором и ПЛИС через память

системы-на-кристалле. Решением может быть увеличение размеров сопроцессора и использование ПЛИС с бóльшим числом логических элементов.

6. Заключение

В данной работе были рассмотрены популярные форматы представления нейронных сетей и открытые инструменты для их исполнения и ускорения. Несколько инструментов было исследовано экспериментально с помощью простых моделей нейронных сетей. Результаты проведенных экспериментов показали, что открытые инструменты способны обрабатывать различные нейронные сети и ускорять их на микро- и графических процессорах. Однако производительность на ПЛИС обычно хуже, за исключением самых простых примеров.

Проведенные эксперименты показали, что применение открытых инструментов вместе с небольшими ПЛИС целесообразно только в системах с жесткими ограничениями на энергопотребление (например, встраиваемых системах) и мягкими требованиями по производительности. Использование открытых инструментов для ускорения нейронных сетей на больших ПЛИС требует дальнейшего исследования.

Дальнейшие исследования возможны и в других областях: разработке специализированных архитектур ускорителей, методах оптимизации нейронных сетей, использованию гетерогенных систем для распределения специфических вычислений, использованию специальных типов нейронных сетей и т.д.

Список литературы / References

[1] TensorFlow framework. Available at: https://www.tensorflow.org, accessed 21.10.2021.
[2] PyTorch framework. Available at: https://pytorch.org, accessed 21.10.2021.
[3] Caffe framework. Available at: https://caffe.berkeleyvision.org, accessed 21.10.2021.
[4] ONNX format. Available at: https://onnx.ai, accessed 21.10.2021.
[5] Lu Y., Gong L. et al. A high-performance FPGA accelerator for sparse neural networks: work-in-progress. In Proc. of the 2017 International Conference on Compilers, Architectures and Synthesis for Embedded Systems Companion, 2017, pp. 1-2.
[6] Sigurbergsson B., Hogervorst T. et al. Sparstition: A Partitioning Scheme for Large-Scale Sparse Matrix Vector Multiplication on FPGA. In Proc. of the IEEE 30th International Conference on Application-specific Systems, Architectures and Processors, 2019, pp. 51-58.
[7] Yang L., He Z., Fan D. A Fully Onchip Binarized Convolutional Neural Network FPGA Implementation with Accurate Inference. In Proc. of the International Symposium on Low Power Electronics and Design, 2018, pp. 1-6.
[8] Chi C., Jiang J. R. Logic Synthesis of Binarized Neural Networks for Efficient Circuit Implementation. In Proc. of the IEEE/ACM International Conference on Computer-Aided Design, 2018, pp. 1-7.
[9] Zhuge C., Liu X. et al. Face Recognition with Hybrid Efficient Convolution Algorithms on FPGAs. In Proc. of the Great Lakes Symposium on VLSI, 2018, pp. 123-128.
[10] Xiao Q., Liang Y. et al. Exploring Heterogeneous Algorithms for Accelerating Deep Convolutional Neural Networks on FPGAs. In Proc. of the 54th Annual Design Automation Conference, 2017, pp. 1-6.
[11] Lu L., Liang Y. SpWA: an efficient sparse Winograd convolutional neural networks accelerator on FPGAs. In Proceedings of the 55th Annual Design Automation Conference, 2018, pp. 1-6.
[12] Alwani M., Chen H. et al. Fused-layer CNN accelerators. In Proc. of the 49th Annual International Symposium on Microarchitecture, 2016, pp. 1-12.
[13] Samragh M., Javaheripi M., Koushanfar F.F. EncoDeep: Realizing Bit-flexible Encoding for Deep Neural Networks. ACM Transactions on Embedded Computing Systems, vol. 19, issue 6, 2020, Article 43, 29 p.
[14] Ding C., Wang S. et al. REQ-YOLO: A Resource-Aware, Efficient Quantization Framework for Object Detection on FPGAs. In Proc. of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2019, pp. 33-42.
[15] Karki A., Keshava C. P. et al. Detailed Characterization of Deep Neural Networks on GPUs and FPGAs. In Proc.of the 12th Workshop on General Purpose Processing Using GPUs, 2019, pp. 12-21.
[16] Mittal S., Umesh S. A survey on hardware accelerators and optimization techniques for RNNs. Journal of Systems Architecture, vol. 112, 2020, article no. 101839.

[17] Kouris A., Venieris S. I., Bouganis C.-S. A throughput-latency co-optimised cascade of convolutional neural network classifiers. In Proc. of the 23rd Conference on Design, Automation and Test in Europe, 2020, pp. 1656-1661.
[18] Lin X., Yin S. et al. LCP: a layer clusters paralleling mapping method for accelerating inception and residual networks on FPGA. In Proc. of the 55th Annual Design Automation Conference, 2018, pp. 1-6.
[19] Keras framework. Available at: https://keras.io, accessed 21.10.2021.
[20] Bücker H. M., Corliss G. et al., editors. Automatic Differentiation: Applications, Theory, and Implementations. Lecture Notes in Computational Science and Engineering, vol. 50, Springer, 2006, 361 p.
[21] Caffe2 framework. Available at: https://caffe2.ai, accessed 21.10.2021.
[22] CNTK framework. Available at: https://github.com/Microsoft/CNTK, accessed 21.10.2021.
[23] CoreML framework. Available at: https://coremltools.readme.io, accessed 21.10.2021.
[24] MXNet framework. Available at: https://mxnet.apache.org, accessed 21.10.2021.
[25] Theano framework. Available at: https://github.com/Theano/Theano, accessed 21.10.2021
[26] Aesara library. Available at: https://github.com/aesara-devs/aesara, accessed 21.10.2021
[27] LegUp 4.0 high-level synthesis tool. Available at: http://legup.eecg.utoronto.ca, accessed 30.12.2020.
[28] Vivado Design Suite. Available at: https://www.xilinx.com/products/design-tools/vivado.html, accessed 21.10.2021.
[29] OpenVINO toolkit. Available at: https://github.com/openvinotoolkit/openvino, accessed 21.10.2021.
[30] Kaldi Speech Recognition Toolkit. Available at: https://kaldi-asr.org, accessed 21.10.2021.
[31] PlaidML framework. Available at: https://github.com/plaidml/plaidml, accessed 21.10.2021.
[32] nGraph compiler stack. Available at: https://github.com/NervanaSystems/ngraph, accessed 21.10.2021.
[33] MACE framework. Available at: https://github.com/XiaoMi/mace, accessed 21.10.2021.
[34] Apache TVM ML compiler framework. Available at: https://tvm.apache.org, accessed 21.10.2021.
[35] VTA: Deep Learning Accelerator Stack. Available at: https://tvm.apache.org/docs/vta/index.html
[36] XLA neural network compiler. Available at: https://www.tensorflow.org/xla, accessed 21.10.2021.
[37] The Julia programming language. Available at: https://julialang.org, accessed 21.10.2021.
[38] JAX differentiation library. Available at: https://github.com/google/jax, accessed 21.10.2021.
[39] Nx multidimensional array representation library. Available at: https://github.com/elixir-nx/nx, accessed 21.10.2021.
[40] LeFlow tool-flow. Available at: https://github.com/danielholanda/LeFlow, accessed 21.10.2021.
[41] Glow ML compiler. Available at: https://ai.facebook.com/tools/glow, accessed 21.10.2021.
[42] hls4ml tool. Available at: https://github.com/fastmachinelearning/hls4ml, accessed 21.10.2021.
[43] Kreinar E. Rfnoc Neural Network Library Using Vivado HLS. In Proc. of the GNU Radio Conference, 2017, 7 p.
[44] NNgen hardware synthesis compiler. Available at: https://github.com/NNgen/nngen, accessed 21.10.2021.
[45] ONNC compilation framework. Available at: https://onnc.ai, accessed 21.10.2021.
[46] Xilinx Vitis AI development stack for AI inference. Available at: https://github.com/Xilinx/Vitis-AI, accessed 21.10.2021.
[47] Xilinx runtime for FPGA. Available at: https://github.com/Xilinx/XRT, accessed 21.10.2021.
[48] FINN framework. Available at: https://xilinx.github.io/finn, accessed 21.10.2021.
[49] Vivado HLS library for FINN. Available at: https://github.com/Xilinx/finn-hlslib, accessed 21.10.2021.
[50] Terasic DE10-Standard board. Available at: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=1081. accessed 21.10.2021.
[51] Terasic DE1-SoC board. Available at: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=836, accessed 21.10.2021.
[52] Zybo Z7-20 board. Available: https://www.xilinx.com/products/boards-and-kits/1-pukio3.html, accessed 21.10.2021.
[53] Quartus Prime Software Suite. Available at: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html, accessed 21.10.2021.
[54] Linux development repository for socfpga. Available at: https://github.com/altera-opensource/linux-socfpga/tree/socfpga-4.14.130-ltsi, accessed 21.10.2021.
[55] Ubuntu Base rootfs. Available at: https://wiki.ubuntu.com/Base, accessed 21.10.2021.
[56] Linux drivers (modules) for Field Programmable Systems on Chip. Available at: http://git.edi.lv/rihards.novickis/FPSoC_Linux_drivers, accessed 21.10.2021.

Информация об авторах / Information about authors

Михаил Сергеевич ЛЕБЕДЕВ – сотрудник ИСП РАН, а также научной лаборатории «Гетерогенные компьютерные системы» РЭУ им. Г.В. Плеханова. Сфера научных интересов: высокоуровневый синтез, методы исследования проектных альтернатив, нейронные сети, цифровая аппаратура, методы верификации цифровой аппаратуры.

Mikhail Sergeyevich LEBEDEV is a specialist at ISP RAS and the «Heterogeneous computer systems» laboratory of Plekhanov RUE. Research interests: high-level synthesis, design space exploration methods, neural networks, digital hardware, hardware verification.

Павел Николаевич БЕЛЕЦКИЙ – сотрудник ИСП РАН. Сфера научных интересов: высокоуровневый синтез, искусственные нейронные сети, ускорение искусственных нейронных сетей, математическое моделирование биологических систем.

Pavel Nikolaevich BELECKY is a specialist of ISP RAS. Research interests: high-level synthesis, artificial neural networks, acceleration of artificial neural networks, mathematical modeling of biological systems.

DOI: 10.15514/ISPRAS-2021-33(6)-13

Weakly Supervised Word Sense Disambiguation Using Automatically Labelled Collections

[1] *A.S. Bolshina, ORCID: 0000-0002-9106-7192<angelina_ku@mail.ru>*
[2] *N.V. Loukachevitch, ORCID: 0000-0002-1883-4121<louk_nat@mail.ru>*
[1] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia
[2] *Research Computing Center Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. State-of-the-art supervised word sense disambiguation models require large sense-tagged training sets. However, many low-resource languages, including Russian, lack such a large amount of data. To cope with the knowledge acquisition bottleneck in Russian, we first utilized the method based on the concept of monosemous relatives to automatically generate a labelled training collection. We then introduce three weakly supervised models trained on this synthetic data. Our work builds upon the bootstrapping approach: relying on this seed of tagged instances, the ensemble of the classifiers is used to label samples from unannotated corpora. Along with this method, different techniques were exploited to augment the new training examples. We show the simple bootstrapping approach based on the ensemble of weakly supervised models can already produce an improvement over the initial word sense disambiguation models.

Keywords: Word sense disambiguation; Russian dataset; RuWordNet.

For citation: Bolshina A.S., Loukachevitch N.V. Weakly supervised word sense disambiguation using automatically labelled collections. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 193-204 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–13

Acknowledgements. This research has been supported by the Interdisciplinary Scientific and Educational School of Lomonosov Moscow State University “Brain, Cognitive Systems, Artificial Intelligence”.

Разрешение неоднозначности на основе псевдоаннотированной коллекции

[1] *А.С. Большина, ORCID: 0000-0002-9106-7192 <angelina_ku@mail.ru>*
[2] *Н.В. Лукашевич, ORCID: 0000-0002-1883-4121 <louk_nat@mail.ru>*
[1] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1
[2] *Научно-исследовательский вычислительный центр МГУ,*
119991, Россия, Москва, Ленинские горы, д. 1., стр. 4

Аннотация. Передовые системы разрешения неоднозначности основаны на обучении с учителем, однако для создания таких моделей требуются большие объемы размеченных данных, которые отсутствуют для большинства языков с ограниченными ресурсами. Для того, чтобы решить проблему недостатка аннотированных данных в русском языке, в данной статье предлагается подход для автоматической разметки значений многозначных слов с использованием ансамбля моделей, базирующихся на слабо контролируемом обучении. Для первичной разметки данных использовался автоматический метод, основанный на концепте однозначных родственных слов. С помощью этих синтетических данных были обучены три модели для разрешения неоднозначности, которые затем применялись в ансамбле для получения значений ключевых многозначных слов. Проведенные

эксперименты показали, что модели, обученные на данных, размеченных предобученными моделями, демонстрируют более высокое качество разрешения неоднозначности. Помимо этого, в статье изучается влияние различных подходов к аугментации текстовых данных на качество предсказаний.

Ключевые слова: автоматическое разрешение неоднозначности; датасеты на русском языке; RuWordNet

Для цитирования: Большина А.С., Лукашевич Н.В. Разрешение неоднозначности на основе псевдоаннотированной коллекции. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 193-204 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(6)–13

Благодарности: Исследование выполнено при поддержке Междисциплинарной научно-образовательной школы Московского государственного университета имени М.В. Ломоносова «Мозг, когнитивные системы, искусственный интеллект».

1. Introduction

The task of Word Sense Disambiguation (WSD) consists in identifying the correct sense of a polysemous word in the context. As with many other NLP tasks, WSD suffers from the problem that is called the knowledge acquisition bottleneck. The recent advances in the field of WSD can be applied only to some languages because obtaining hand-crafted sense-labelled training collections is very expensive in terms of time and extensive human efforts. The low-resource languages do not have access to the large labelled collections that are necessary for training current state-of-the-art supervised models. And that hinders the development of the different applications closely related to the WSD task, for example, semantic text analysis, knowledge graph construction, machine translation, question answering, etc.

In recent years to address these challenges, practitioners turn to weak supervision that implies training models using data with imperfect labels, that can be obtained with some user-defined heuristics, external knowledge bases, other classifiers etc. Various methods of automatic acquisition of training samples have been invented in the field of WSD. In our research we utilize the method to automatically generate and label training collections with the help of monosemous relatives, that is a set of unambiguous words (or phrases) related to particular senses of a polysemous word. The labels obtained with the help of this approach were used to train three different weakly supervised WSD models: logistic regression with the deep representations from ELMo [1] language model as features, fine-tuned BERT [2] model and BERT model trained on context-gloss pairs.

In this article, we propose an algorithm based on the ensemble of weakly supervised WSD models that can be used to label raw texts and, thus, reduce human efforts to annotation. The additional data provided by the algorithm was used to re-train original models, and the experiments showed that it enhanced the initial models' performance. Moreover, leveraging different augmentation techniques we were also able to improve upon the classification results.

The paper is organized as follows. In section 2 we review the related work. Section 3 is devoted to the data description. The fourth section describes the method applied to automatically generate and annotate training collections. The models and augmentation techniques are presented in the fifth section. In the sixth section, we describe an algorithm based on the weighted probabilistic ensemble of the WSD models used to predict sense labels and in Section 7 we demonstrate the results obtained by three different models. Concluding remarks are provided in the eighth section.

2. Related work

To overcome the limitations, that are caused by the lack of annotated data, several methods of generating and harvesting large train sets have been developed. There exist many techniques based on different kinds of replacements, which do not require human resources for tagging. The most popular method is that of monosemous relatives [3]. Usually, WordNet [4] is used as a source for such relatives. WordNet is a lexical-semantic resource for the English language that contains a

description of nouns, verbs, adjectives, and adverbs in the form of semantic graphs. All words in those networks are grouped into sets of synonyms that are called synsets.

Monosemous relatives are those words or collocations that are related to the target ambiguous word through some connection in WordNet, but they have only one sense, i.e. belong only to one synset. Usually, synonyms are selected as relatives but in some works hypernyms and hyponyms are chosen [5]. Some researchers replace the target word with named entities [6], some researchers substitute it with meronyms and holonyms [7]. In the work [8] distant relatives (including distant hypernyms and hyponyms) were used; the procedure of training contexts selection was based on the distance to a target word and the type of the relation connecting the target sense and a monosemous relative.

Multilingual resources such as parallel corpora are also a valuable source of information that can be used to generate training collections for the WSD task [9]–[12]. Other methods of automatic annotation of training collections for WSD exploit knowledge bases like Wikipedia and Wiktionary [13]–[16]. In bootstrapping approach, the classifier relies on a small number of labelled seed instances, then a set of raw samples with the highest confidence is annotated using this model and utilized to retrain the model. The whole cycle of this procedure is repeated until the desired number of samples is labelled or some benchmark in performance is reached [17]–[19].

It is clear, that the above-mentioned methods cannot guarantee correct labelling of the samples, however, such imperfect data can still be used in weak supervision. This strategy is used extensively for named entity recognition [20], relation extraction [21], [22], entity linking [23] and text classification [24]. As weak supervision can introduce different types of noise into a model, in our research to infer the sense label of the unannotated sample, we combined the predicted class probabilities of the three weakly supervised models alongside uncertainty estimation.

Nowadays, the greater part of the WSD systems is based on neural networks. Recent studies have shown the effectiveness of contextualized word representations for the WSD task [25], [26]. The most widely used deep contextualized embeddings are ELMo [1] and BERT [2]. Let us briefly overview some of the state-of-the-art approaches in WSD. The system based on Transformer encoders, BERT contextualized word embeddings and sense vocabulary compression methods were introduced in [27]. The most significant feature of the algorithm EWISE is “predicting over a continuous sense embedding space as opposed to a discrete label space” [28]. The system EWISER [29] builds upon EWISE. But to better predict unseen words, the information about concepts and their relations from WordNet was added to this neural architecture. The work [30] focuses on exploring sparse contextualized word representations as a solution to the task of fine-grained WSD. The work [31] introduces the bi-encoder model for WSD. Context and gloss encoders are independent of each other and are initialized with BERT, but their output representations are combined to predict sense labels.

In the current research, we implemented a simple logistic regression model with ELMo representations as features, also we employed BERT for fine-tuning and sentence pair classification task between the sentence with a target polysemous word and the gloss related to one of its senses. These models are described in more detail in Section 5. We used automatically generated training collections to train these classifiers. Then with the help of these models, we annotated unlabeled data with noise labels and used it to train new WSD models.

3. Data

In our research as an underlying semantic network, we exploit Russian thesaurus RuWordNet [32]. It is a semantic network for Russian that has a WordNet-like structure. In total it contains 111.5 thousand words and word combinations for the Russian language. RuWordNet was used to extract semantic relations (e.g., synonymy, hyponymy etc.) between a target sense of a polysemous word and all the words (or phrases) connected to it, including those linked via distant paths. The sense inventory was also taken from this resource. RuWordNet contains 29297 synsets for nouns, 63014 monosemous and 5892 polysemous nouns. In this research, we consider only ambiguous nouns.

We utilized two types of corpora in the research. A news corpus consists of news articles harvested from various news sources. The texts have been cleaned from HTML elements or any markup. Another corpus consists of the several segments of Taiga corpus [33], which are compiled from news articles: Lenta.ru, Interfax, Komsomolskaya Pravda, Russian Magazines Hall, Fontanka.ru. We exploit these two corpora for extracting the sentences with target polysemous words and subsequent labelling by the ensemble of models. Moreover, the news corpus was exploited for the training word2vec model necessary for the algorithm of automatic generation of training collections.

Table 1. Quantitative characteristics of the target polysemous words and their senses

Target word and its sense	**Number of validation samples**	**Number of glosses**
аниматор$_0$ “a cartoonist”	29	11
аниматор$_1$ “an entertainer”	28	4
барометр$_0$ “a barometer”	24	17
барометр$_1$ “an indicator”	24	5
болячка$_0$ “an illness”	21	9
болячка$_1$ “a wound”	21	9
графит$_0$ “graphite”	24	5
графит$_1$ “a pencil”	17	6
дичь$_0$ “a fowl”	31	7
дичь$_1$ “nonsense”	17	9
зайчик$_0$ “sunbeam”	11	6
зайчик$_1$ “a bunny”	14	5
зародыш$_0$ “an embryo”	18	6
зародыш$_1$ “beginning”	10	7
калейдоскоп$_0$ “in the thick of it”	12	6
калейдоскоп$_1$ “kaleidoscope”	14	6
колыбель$_0$ “a crib”	16	5
колыбель$_1$ “the place of origin”	13	6
колокольчик$_0$ “a bluebell”	15	5
колокольчик$_1$ “a bell”	16	7

There are two variants of the WSD task: lexical sample and all-words. The former consists in disambiguating a small pre-selected set of polysemous words, the latter, on the contrary, implies predicting sense for each polysemous word in a text. In our experiments, we perform lexical sample sense disambiguation, which is why we chose several polysemous words, for which the WSD models of different types would be implemented, in advance. The selection criterion was as follows: the word should occur in the corpora at least 500 times and in no less than 200 documents. To evaluate the models, for each sense of the target polysemous words, we manually labelled small validation datasets compiled from the news articles from Wikinews.

Glosses are widely used in the field of WSD: for example, they can be utilized directly to disambiguate a sample [34] or can serve as a weakly supervised signal in models [31], [35]–[37]. The work [38] demonstrated that word definitions and examples of use can be used to augment training data and even boost the performance of a WSD system. For that reason, for each sense of a target polysemous word, we collected a small set of dictionary definitions and examples taken from dictionary entries. This data is intended to be added to a training collection along with the texts labelled by the ensemble of WSD models, and ultimately utilized in retraining.

The list of the selected polysemous words, the number of annotated samples and glosses for each sense is given in Table 1.

4. Method of automatic labelling of training collections

For the preliminary annotation of the training data, we employed the method described in [39], that is based on the concept of monosemous relatives. This approach for collecting a training corpus is based on the substitution: for every polysemous word we select appropriate monosemous relatives, then in a text, the occurrences of these relatives are substituted by the target polysemous word and these instances are labelled with a sense tag of a monosemous relative.

The findings from the research [39] showed, that the utilization of distant relatives (e.g., cohyponyms) along with synonyms, hyponyms and hypernyms enables a wider coverage of the target polysemous words in a training collection. In our research, the distance between the target sense of the polysemous word and its candidate monosemous relatives can reach up to four steps in the semantic graph.

Not all monosemous relatives are suitable as a representation of a target word polysemous word. To ensure, that the contexts with monosemous relatives extracted from a corpus will serve as good training samples for the target sense, we utilized a custom word2vec embedding model trained on the same corpus from which the contexts are retrieved. With the help of this model, we compute the similarity between the contexts, in which the candidate monosemous relative occur, and the words located close to the target polysemous word (within two steps from the target word).

For example, the selected monosemous relatives for the word болячка$_1$ “a wound” are "ссадина, волдырь, мозоль, оспина, прыщ, прыщик, струп" *(abrasion, blister, callus, pockmark, pimple, little pimple, scab*). For the word болячка$_0$ “an illness” the monosemous realtives are as follows: "болезнь, ангина, диабет, воспаление, бронхит, обморок, травматизм, астма, артроз" (*disease, sore throat, diabetes, inflammation, bronchitis, fainting, injury, asthma, arthrosis).*

This approach has already been applied to the subset of words from the RUSSE’2018 [40] evaluation dataset. The experiments showed that the models trained on the automatically generated collections can obtain the quality of disambiguation comparable to the models trained on the manually labelled data. In this research, we want to explore the application of the proposed method to other words with different level of ambiguity (see Section 3 for details).

5. Models

As it has already been said, in our research we employ three diverse supervised WSD models: two of them are based on ruBERT pretrained representations [41] released by DeepPavlov and the other employs RusVectōrēs [42] ELMo model trained on lemmatized Tayga corpus. The first model is a fine-tuned BERT with a sequence classification head: a linear layer on top of the concatenated target token representations from the last four hidden layers of the pre-trained transformer.

The second model is based upon the ideas from [36] and [43]: we utilized context-gloss pair with weak supervision for sentence pair classification task performed by the BERT model. The first element of a pair is a sentence with a target polysemous word; in each sentence, we put the target word in quotation marks as a weak supervised signal. The second element of a pair is a gloss definition of one of the senses of the ambiguous word. At the beginning of each gloss, we put the target word as a weak supervised signal. The two parts of training samples are concatenated with the special BERT symbol *[SEP]*, and are marked as positive ones only if the definition corresponds to the correct sense. Lemmatized context-gloss pairs from the automatically generated training collection are presented in Table 2. The experiments with Russian WSD models [25], [38] demonstrated that lemmatized training improves the performance of the models. For that reason, in all our models, we used lemmatized training samples, as can clearly be seen from Table 2.

Following [25], we also utilized in the experiments a simple logistic regression classifier, that uses ELMo representations as features. Additionally, [38] showed that the optimal way to use RusVectōrēs ELMo embeddings for the WSD task is to extract embedding solely for a target polysemous word, thus, for our experiments, we extracted the single vector of the target word from the ELMo top layer.

It should be noted, that in our research we investigate the performance of the monolingual WSD models trained on the automatically generated training collections and on the pseudo-labelled ones, that is why we do not explore multilingual WSD models. However, the results obtained in the described experiments can be used as a baseline of comparison for future work.

Table 2. Automatically created and labelled context-gloss pairs

Training sample	**Label**
кожа покраснение "болячка" припухлость круг глаз воспользоваться консилер [SEP] болячка : болезнь	0
кожа покраснение "болячка" припухлость круг глаз воспользоваться консилер [SEP] болячка : болезненный образование на тело	1
выключатель адреналин необходимый создание ингибитор "болячка" бета блокатор [SEP] болячка : болезненный образование на тело	0
выключатель адреналин необходимый создание ингибитор "болячка" бета блокатор [SEP] болячка : болезнь	1

6. Experimental design

Automatically generated data is noisy, and it is clear that models trained on such imperfect data may be prone to errors. In this article, we propose the solution to mitigate this problem common to weakly supervised systems. First, we train three models described in the previous section. Second, we utilize them to predict sense tag for each target sample in the unlabelled corpus. The core idea of our experiment is to use the ensemble of the three types of models to predict sense labels to the raw texts, taking into account the models' uncertainty level. These texts would then constitute the new training dataset. Finally, all the above-mentioned classifiers would be retrained on this pseudo-annotated data.

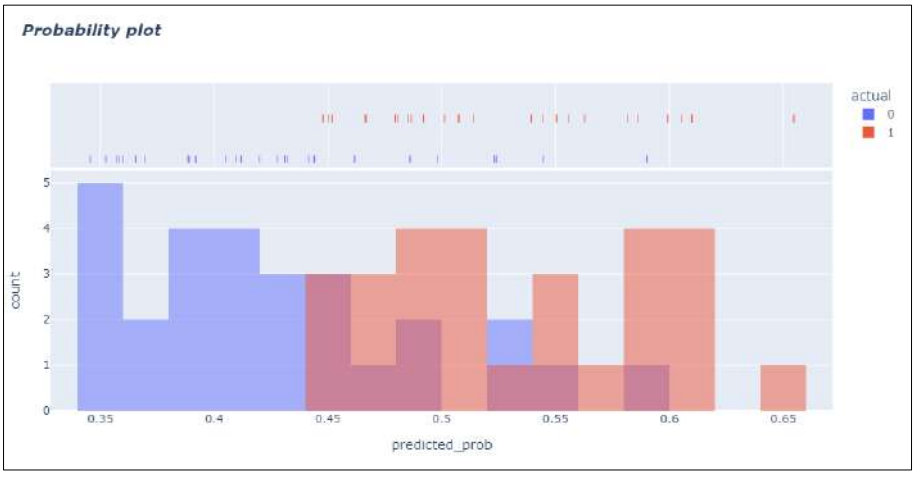


Fig. 1. The predicted probabilities for the validation samples with the word "аниматор": the logistic regression model

First of all, for each classifier, we defined the range of probabilities where it is uncertain. These thresholds help to filter poor classifiers' predictions. A fine-tuned BERT model and a logistic regression classifier output a single probability of a class. We, therefore, employed the validation

dataset, and the models' probability estimations, in particular, to identify zones where the predictions get confused the most.

In Fig. 1, the area where the model makes most of the mistakes could be seen. Hence, in the case of this model, we would not trust the predictions that fall into the probability range from 0.45 to 0.6.

To predict the sense label with the help of the context-gloss pair model, one needs to compare the probabilities of all the context-gloss pairs available for this or that target word. To derive the criteria for this type of model, for each sense of the target word we analyzed the differences between the probabilities of the correct and incorrect class predicted for the validation samples. If this difference is more than 0 then the model predicted the right label. Thus, in our research, the 0.25-quantile of the positive values of difference is considered to be a threshold for the context-gloss pair model.

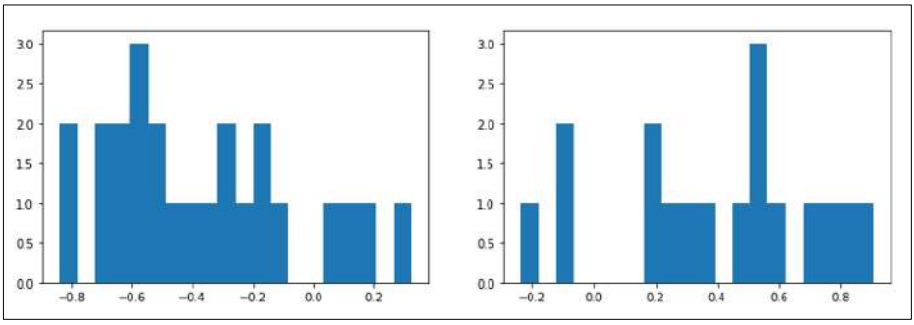


Fig. 2. The differences in the predicted probabilities for the validation samples with the word "графит": "graphite" and "a pencil" senses, respectively.

According to the data presented in Fig. 2, the threshold for the first class ("graphite") amounted to 0.1, whereas for the second class, this value is 0.3. So, if the difference between the classifier predictions is less than these values, we discard these probability values.

Thus, we have defined the threshold values for each of the WSD classifiers. The probability estimations, that does not meet the specified conditions, are discarded. To obtain the final class label from the probabilities, that comply with the requirements, we apply a weighting function to the models' output. There are various types of weighting schemes but we concentrate on the ones, where "rather than a single weight w_j, a separate weight is assigned to each class w_{ij}. This weight is set to be the proportion of cases correct for that class on the training data" [44]. Therefore, in our system, each base classifier prediction is multiplied by the precision value of this or that class obtained during the evaluation of the model on the validation set. Then all the weighted outcomes are summed, and the index of the maximum probability is returned as the final sense label for the sample with the target word. This weighting scheme allows us to rely on the predictions of other classifiers when some of them are not precise enough.

To boost the performance of the ensemble, we resorted to the principle "One sense per discourse" [45]: "if a polysemous word such as sentence appears two or more times in a well-written discourse, it is extremely likely that they will all share the same sense". From the news corpus and the segments of Tayga mentioned in Section 3, we extracted all the texts with the target polysemous words, in which they appeared more than twice. With the help of our scheme described above, for each occurrence of the target word, we predicted the class label. The final class label for all the samples, that constitute the text, is chosen according to the majority voting: we selected the class label that has more than half the votes. If this condition is not met, the label is determined by the maximum value of mean class probabilities.

Another constituent part of our experiment is augmentations. We have already described the augmentations with dictionary definitions and examples of use in Section 3. In addition, we applied easy data augmentation techniques from [46] to the samples labelled with the help of the ensembles. The maximum number of examples annotated by this strategy is 804 (total for the word "колокольчик"), which is not enough to train the models based on BERT. Consequently, we adapted

the original implementation of augmentation techniques for the WSD task in Russian: added the extraction of the synonyms from RuWordNet and imposed the limitation on the transformations of the original sentence (they should not involve the target polysemous word). The number of generated augmented sentences per original sentence was set to 6, this augmented data was used solely for retraining of the BERT-based models.

7. Results

In this section, we present the results of our experiments. It should be specially noted, that the evaluation on the validation dataset was performed with two different context windows: win=1 implies that the context includes one sentence before and after the sentence with the target word; when using win=0, we took only the sentence with the target word. In Table 3 we demonstrate the averaged f1-scores obtained with the models trained on the automatically generated training collections and the data pseudo-labelled by the ensemble of models. Table 4 contains the F1-scores for the models retrained on the new pseudo-annotated data with "One sense per discourse" assumption. In these tables, by (1) we denoted ELMo LogReg model, (2) is Fine-tuned BERT and (3) is Context-gloss pair BERT. We validated the models on the four variants of datasets: (a) is the dataset compiled without "One sense per discourse" principle and without dictionary definitions augmentation, (b) was composed without "One sense per discourse" principle but with dictionary definitions augmentation, (c) was created with "One sense per discourse" principle and without dictionary definitions augmentation, (d) was created with "One sense per discourse" principle and with dictionary definitions augmentation.

Table 3. Averaged classification results for the WSD models (F1-score)

Dataset	**ELMo LogReg**	**Fine-tuned BERT**	**Context-gloss pair BERT**
Dataset automatically labelled with the monosemous relatives approach	0.85	0.81	0.79
(a)	0.86	0.84	**0.87**
(b)	**0.86**	0.85	**0.86**
(c)	**0.87**	0.84	0.86
(d)	0.87	**0.88**	0.87

Table 4. Classification results for the WSD models trained on pseudo-labelled data with "One sense per discourse assumption" (F1-score).

Target word	**(c)**			**(d)**		
	(1)	**(2)**	**(3)**	**(1)**	**(2)**	**(3)**
аниматор win=1	0.74	0.7	0.78	0.79	**0.89**	0.85
аниматор win=0	0.8	0.7	0.77	0.79	0.8	0.77
барометр win=1	0.96	0.93	0.88	**0.98**	0.93	0.88
барометр win=0	0.94	0.91	0.87	0.92	0.94	0.9
болячка win=1	0.73	0.69	0.61	0.76	0.68	0.7
болячка win=0	0.76	0.69	0.74	**0.77**	0.73	0.75
графит win=1	0.6	0.59	0.78	0.65	**0.79**	0.74
графит win=0	0.65	0.59	0.72	0.63	0.74	0.74
дичь win=1	**0.97**	**0.97**	0.96	0.97	**0.97**	0.96
дичь win=0	0.9	0.9	0.95	0.87	**0.97**	0.95
зайчик win=1	**1**	**1**	0.96	**1**	**1**	0.96
зайчик win=0	**1**	**1**	0.98	**1**	**1**	0.98
зародыш win=1	0.95	**1**	0.96	0.95	**1**	0.96
зародыш win=0	0.86	0.95	0.88	0.86	0.95	0.92

калейдоскоп win=1	0.85	0.74	0.79	0.85	0.74	0.79
калейдоскоп win=0	**0.88**	0.74	0.84	**0.88**	0.77	0.87
колыбель win=1	**0.93**	0.87	0.86	**0.93**	0.87	0.89
колыбель win=0	0.9	0.9	**0.93**	0.9	0.9	**0.93**
колокольчик win=1	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**
колокольчик win=0	0.94	0.93	**0.97**	0.94	**0.97**	**0.97**
averaged f1 for win=1	0.87	0.85	0.86	**0.89**	**0.89**	0.87
averaged f1 for win=0	0.86	0.83	0.87	0.86	**0.88**	**0.88**

The results of the models retrained on the new texts labelled with the ensembles show that this procedure improves the overall performance of the models. In some cases, the retraining greatly increases the F1-score of a classifier, for example, the maximum f1-score 0.75 for the word "аниматор" (win=1) was achieved with the initial context-gloss pair BERT model, after the retraining on the pseudo-labelled data the score rose to 0.85. Sometimes the effect is less clearly defined, e.g. the results of the logistic regression model for the word "барометр" slightly change across different data modifications. However, sometimes we can see that the quality of the models trained on the pseudo-labelled data can be worse than the performance of the initial models: the LogReg classifiers trained for the word "болячка" on the data without using “One sense per discourse assumption” have lower f1-score than the initial classifiers.

In most cases, the models trained on the data that was obtained employing the “One sense per discourse” principle show higher results on the validation dataset. But there are several cases when this was not so: for example, the BERT models’ results for the word "калейдоскоп".

As for the dictionary definitions augmentations, the data shows that this additional data either has no effect on the performance score or, like in most of the cases, the f1-score improves. We also can see, that the window size has a varying effect on the performance of the WSD models, and its impact is yet to be investigated. Moreover, the data demonstrates that there is no clear trend in the type of the model that performs best: in some cases, ELMo has the highest f1-score ("барометр" (win=1), f1=0.98), sometimes it is fine-tuned BERT ("аниматор" (win=1), f1=0.89), context-gloss pair BERT outperformed all other models for the word "колыбель" (win=0), f1=0.93.

The experiments also proved that all the words are different in the degree of ambiguity. Some words have a very high f1-score, which means that they are easier to be disambiguated, for example, the word "зайчик", whose senses are well-differentiated. In contrast, the word "графит" has the metonymic type of the polysemy, its senses are connected as "the material-the product made of it". This word has a lower f1-score compared to the other words because its senses are hard to be identified.

The aim of our experiment is not to find the best WSD method in general. Rather, the goal is to find the method that improves the models trained on the data with weak labels. Our experiments proved that gradual retraining of the WSD models on the newly labelled data, i.e. bootstrapping, can enhance the overall performance of classification. Moreover, the proposed probabilistic ensemble weighting strategy can be utilized as an aid to manual sense annotation, for example, in the active learning environment.

8. Conclusion

In this work, we introduced the probabilistic ensemble weighting scheme, which is aimed at producing less noisy training data for the WSD classifiers. We proved that this strategy is robust in cases with automatically generated training collections, especially because we added an uncertainty estimation component. Moreover, additional data generated by the augmentation techniques have been shown to aid model performance.

The experiments demonstrated that retraining the models on the new data labelled utilizing the ensembles improved upon the initial results of the WSD models. The continuous retraining of the models on the new sets of samples can further boost the performance of lexical ambiguity resolution. Also, the probabilistic ensemble weighting scheme can be used to facilitate the efforts to manually label training data.

References

[1]. Peters M. E., Neumann M. et al. Deep contextualized word representations. In Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2018, pp. 2227–2237.

[2]. Devlin J., Chang M.-W. et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Association for Computational Linguistics, 2019, pp. 4171-4186.

[3]. Leacock C., Chodorow M., Miller G.A. Using corpus statistics and WordNet relations for sense identification. Computational Linguistics, vol. 24, no. 1, 1998, pp. 147–165.

[4]. Miller G. A. WordNet: a lexical database for English. Communications of the ACM, vol. 38, no. 11, 1995, pp. 39–41.

[5]. Przybyła P. How big is big enough? Unsupervised word sense disambiguation using a very large corpus. arXiv preprint arXiv:1710.07960, 2017.

[6]. Mihalcea R., Moldovan D.I. An Iterative Approach to Word Sense Disambiguation. In Proc. of the Thirteenth International Florida Artificial Intelligence Research Symposium Conference, 2000, pp. 219-223.

[7]. Yuret D. KU: Word sense disambiguation by substitution. In Proc. of the Fourth International Workshop on Semantic Evaluations (SemEval-2007), 2007, pp. 207-214.

[8]. Martinez D., Agirre E., Wang X. Word relatives in context for word sense disambiguation. In Proc.of the Australasian Language Technology Workshop, 2006, pp. 42-50.

[9]. Taghipour K., Ng H.T. One million sense-tagged instances for word sense disambiguation and induction. In Proc. of the Nineteenth Conference on Computational Natural Language Learning, 2015, pp. 338–344.

[10]. Otegi A., Aranberri N. et al. QTLeap WSD/NED corpora: Semantic annotation of parallel corpora in six languages. In Proc. of the Tenth International Conference on Language Resources and Evaluation (LREC'16), 2016, pp. 3023–3030.

[11]. Bovi C.D., Camacho-Collados J. et al. Eurosense: Automatic harvesting of multilingual sense annotations from parallel text. In Proc. of the 55th Annual Meeting of the Association for Computational Linguistics (vol. 2: Short Papers), 2017, pp. 594–600.

[12]. Hauer B., Kondrak G. et al. Semi-Supervised and Unsupervised Sense Annotation via Translations. arXiv preprint arXiv:2106.06462, 2021.

[13]. Henrich V., Hinrichs E., Vodolazova T. WebCAGe–A Web-harvested corpus annotated with GermaNet senses. In Proc. of the 13th Conference of the European Chapter of the Association for Computational Linguistics, 2012, pp. 387-396.

[14]. Saif A., Omar N. et al. Building Sense Tagged Corpus Using Wikipedia for Supervised Word Sense Disambiguation. Procedia Computer Science, vol. 123, 2018, pp. 403-412.

[15]. Raganato A., Bovi C.D., Navigli R. Automatic Construction and Evaluation of a Large Semantically Enriched Wikipedia. In Proc. of the Twenty-Fifth International Joint Conference on Artificial Intelligence, 2016, pp. 2894-2900.

[16]. Scarlini B., Pasini T., Navigli R. Just "OneSeC" for producing multilingual sense-annotated data. In Proc. of the 57th Annual Meeting of the Association for Computational Linguistics, 2019, pp. 699-709.

[17]. Mihalcea R. Co-training and self-training for word sense disambiguation. In Proc. of the Eighth Conference on Computational Natural Language Learning (CoNLL-2004), 2004, pp. 33-40.
[18]. Pham T.P., Ng H.T., Lee W.S. Word sense disambiguation with semi-supervised learning. Lecture Notes in Computer Science, vol. 3406, 2005, pp. 238-241.
[19]. Khapra M. M., Joshi S. et al. Together we can: Bilingual bootstrapping for WSD. In Proc. of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, 2011, pp. 561-569.
[20]. Lison P., Barnes J., Hubin A. skweak: Weak supervision made easy for NLP. In Proc. of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations, 2021, pp. 337-346.
[21]. Lin Y., Shen S. et al. Neural relation extraction with selective attention over instances. In Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (vol. 1: Long Papers), 2016, pp. 2124-2133.
[22]. Li Z., Hu F. et al. Selective kernel networks for weakly supervised relation extraction. CAAI Transactions on Intelligence Technology, vol. 6, no. 2, 2021, pp. 224-234.
[23]. Le P., Titov I. Boosting entity linking performance by leveraging unlabeled documents. In Proc. of the 57th Annual Meeting of the Association for Computational Linguistics, 2019, pp. 1935-1945.
[24]. Wang Y., Sohn S. et al. A clinical text classification paradigm using weak supervision and deep representation. BMC medical informatics and decision making, vol. 19, no. 1, 2019, pp. 1-13.
[25]. Kutuzov A., Kuzmenko E. To lemmatize or not to lemmatize: how word normalisation affects ELMo performance in word sense disambiguation. arXiv preprint arXiv:1909.03135, 2019.
- Wiedemann G., Remus S. et al. Does BERT Make Any Sense? Interpretable Word Sense Disambiguation with Contextualized Embeddings. In Proc. of the 15th Conference on Natural Language Processing (KONVENS 2019): Long Papers, 2019, pp. 161-170.

[26]. Vial L., Lecouteux B., Schwab D. Sense vocabulary compression through the semantic knowledge of wordnet for neural word sense disambiguation. arXiv preprint arXiv:1905.05677, 2019.
[27]. Kumar S., Jat S. et al. Zero-shot word sense disambiguation using sense definition embeddings. In Proc. of the 57th Annual Meeting of the Association for Computational Linguistics, 2019, pp. 5670-5681.
[28]. Bevilacqua M., Navigli R. Breaking through the 80% glass ceiling: Raising the state of the art in Word Sense Disambiguation by incorporating knowledge graph information. In Proc. of the 58th Annual Meeting of the Association for Computational Linguistics, 2020, pp. 2854-2864.
[29]. Berend G. Sparsity Makes Sense: Word Sense Disambiguation Using Sparse Contextualized Word Representations. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 8498-8508.
[30]. Blevins T., Zettlemoyer L. Moving down the long tail of word sense disambiguation with gloss-informed biencoders. arXiv preprint arXiv:2005.02590, 2020.
[31]. Loukachevitch N. V., Lashevich G., Gerasimova A. A., Ivanov V. V., Dobrov B. V. Creating Russian wordnet by conversion. In Computational Linguistics and Intellectual Technologies: Papers from the Annual Conference Dialogue, 2016, pp. 405-415.
[32]. Shavrina T., Shapovalova O. To the methodology of corpus construction for machine learning: “Taiga” syntax tree corpus and parser. In Proc. of “CORPORA-2017” International Conference, 2017, pp. 78-84.
[33]. Lesk M. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In Proc. of the 5th Annual International Conference on Systems Documentation, 1986, pp. 24-26.
[34]. Luo F., Liu T. et al. Incorporating glosses into neural word sense disambiguation. arXiv preprint arXiv:1805.08028, 2018.
[35]. Huang L., Sun C. et al. GlossBERT: BERT for word sense disambiguation with gloss knowledge. arXiv preprint arXiv:1908.07245, 2019.
[36]. Loureiro D., Jorge A. Language modelling makes sense: Propagating representations through WordNet for full-coverage word sense disambiguation. arXiv preprint arXiv:1906.10007, 2019.
[37]. Bolshina A., Loukachevitch N. Exploring the limits of word sense disambiguation for Russian using automatically labelled collections. In Proc. of the Linguistic Forum 2020: Language and Artificial Intelligence (LFLAI), 2020, 14 p.
[38]. Bolshina A., Loukachevitch N. Generating training data for word sense disambiguation in Russian. In Computational Linguistics and Intellectual Technologies: Papers from the Annual International Conference “Dialogue”, 2020, pp. 119-132.

[39]. Panchenko A., Lopukhina A., Ustalov D., Lopukhin K., Arefyev N., Leontyev A., Loukachevitch N. RUSSE'2018: A Shared Task on Word Sense Induction for the Russian Language. In Computational Linguistics and Intellectual Technologies: Papers from the Annual International Conference "Dialogue", 2018, pp. 547–564.
[40]. Kuratov Y., Arkhipov M.Y. Adaptation of deep bidirectional multilingual transformers for Russian language. arXiv preprint arXiv: 1905.07213, 2019.
[41]. Kutuzov A., Kuzmenko E. WebVectors: a toolkit for building web interfaces for vector semantic models. Communications in Computer and Information Science, vol. 661, 2016, pp. 155-161.
[42]. Kohli H. Transfer learning and augmentation for word sense disambiguation. In Advances in Information Retrieval, Springer, 2021, pp. 303-311.
[43]. Large J., Lines J., Bagnall A. A probabilistic classifier ensemble weighting scheme based on cross-validated accuracy estimates. Data Mining and Knowledge Discovery, vol. 33, no. 6, 2019, pp. 1674-1709.
[44]. Gale W. A., Church K. W., Yarowsky D. One sense per discourse. In Proc. of the Workshop on Speech and Natural Language, 1992, pp. 233-237.
[45]. Wei J., Zou K. EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In Proc. of the Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 2019, pp. 6383-6389.

Информация об авторах / Information about authors

Ангелина Сергеевна БОЛЬШИНА – аспирант кафедры теоретической и прикладной лингвистики. Сфера научных интересов: автоматическая обработка текстов, автоматическое разрешение неоднозначности, глубокое обучение.

Angelina Sergeevna BOLSHINA – PhD student. Research interests: natural language processing, word sense disambiguation, deep learning.

Наталья Валентиновна ЛУКАШЕВИЧ – доктор технических наук, ведущий научный сотрудник. Сфера научных интересов: автоматическая обработка текстов, онтологии, анализ тональности.

Natalia Valentinovna LOUKACHEVITCH – Doctor of Technical Sciences, leading researcher. Research interests: natural language processing, ontologies, sentiment analysis.

DOI: 10.15514/ISPRAS-2021-33(6)-14

Активное обучение и перенос знаний в задаче сегментации изображений документов

[1,2] *Д.М. Киранов, ORCID:0000-0002-3507-3803 <kiranov.dm@ispras.ru>*
[1] *М.А. Рындин, ORCID:0000-0002-7504-3975 <mxrynd@ispras.ru>*
[1] *И.С. Козлов, ORCID:0000-0002-0145-1159 <kozlov-ilya@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2] *Московский физико-технический институт,*
141701, Россия, Московская область, г. Долгопрудный, Институтский пер., 9

Аннотация. В данной работе исследуется эффективность классических подходов активного обучения в задаче сегментации изображений документов с целью уменьшения обучающей выборки. Приводится свой модифицированный подход выбора изображений для разметки и последующего обучения. Результаты, полученные с помощью активного обучения, сравниваются с переносом знаний, использующим полностью размеченные данные. Также исследуется, как предметная область обучающего набора, на котором инициализируется модель для переноса знаний, влияет на последующее дообучение модели.

Ключевые слова: активное обучение; перенос знаний; сегментация изображений

Для цитирования: Киранов Д.М., Рындин М.А., Козлов И.С. Активное обучение и перенос знаний в задаче сегментации изображений документов. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 205-216. DOI: 10.15514/ISPRAS–2021–33(6)–14

Active learning and transfer learning for document segmentation

[1,2] *D.M. Kiranov, ORCID:0000-0002-3507-3803 <kiranov.dm@ispras.ru>*
[1] *M.A. Ryndin, ORCID:0000-0002-7504-3975 <mxrynd@ispras.ru>*
[1] *I.S. Kozlov, ORCID:0000-0002-0145-1159 <kozlov-ilya@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2] *Moscow Institute of Physics and Technology,*
9, Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russia

Abstract. In this paper, we investigate the effectiveness of classical approaches of active learning in the problem of segmentation of document images in order to reduce the training sample. A modified approach to the selection of images for marking and subsequent training is presented. The results obtained through active learning are compared to transfer learning using fully labeled data. It also investigates how the subject area of the training set, on which the model is initialized for transfer learning, affects the subsequent additional training of the model.

Keywords: active learning; transfer learning; image segmentation

For citation: Kiranov D.M., Ryndin M.A., Kozlov I.S. Active learning and transfer learning for document segmentation. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 205-216 (in Russian). DOI: 10.15514/ISPRAS–2020–33(6)–14

1. Введение

При анализе структурированных документов, таких как научные статьи, правовые акты, полезным первым шагом для последующего анализа является выделение сложных структурных элементов: текстовых блоков, таблиц, списков, графиков и названий. Разметка изображений для обучения – времяемкое занятие, кроме того задачи сегментирования и обнаружения объектов в некоторых узкоспециализированных предметных областях могут требовать специфичных знаний для корректной аннотации изображений, что ограничивает круг лиц, способных осуществить разметку для обучения. Таким образом, ограниченный человеческий ресурс требует от нас минимизации числа обучающих примеров.

Кроме того, не все изображения являются одинаково полезными для обучения; к примеру, в отсканированных документах объект-текст присутствует почти на каждом изображении, в то время как объект-список встречается не так часто, и, стало быть, обучаясь на нем, модель может извлечь больше полезной информации. Отсюда, в условии минимизации числа обучающих примеров естественным образом возникает вопрос о получении тех примеров, обучившись на которых, модель могла бы лучше всего справляться с поставленной задачей.

В данной работе рассматривается подход, в котором примеры для обучения выбираются с помощью активного обучения. Активное обучение (Active Learning) – это раздел машинного обучения, в котором модель может обращаться к оракулу для разметки выбранных неразмеченных данных [1]. Основной целью активного обучения является получение высокой точности модели на новых примерах с минимальным возможным количеством обращений к оракулу (подробнее см. разд. 2). Главной трудностью при этом является выбор наиболее информативных примеров для обучения.

Классическим подходом отбора примеров для разметки является «выбор по степени неуверенности». К основным мерам неуверенности относятся:

- максимальный отступ [2]

$$\varphi(x)_{margin} = \left[1 - \left(\max_{c_1 \in K} p(c_1|x) - \max_{c_2 \in K \setminus c_1} p(c_2|x)\right)\right]^2;$$

- максимальная энтропия [3]

$$\varphi(x)_{entropy} = -\sum_{c \in K} p(c|x) \log p(c|x);$$

- минимальная уверенность [4]

$$\varphi(x)_{\min\_conf} = 1 - \max_{c_1 \in K} p(c_1|x).$$

1.1 Активное обучение в задачах обнаружения объектов и сегментации

На сегодняшний день существует немало статей [5-9], посвященных исследованию подходов активного обучения в задачах обнаружения объектов и сегментации изображений, которые показали свою эффективность. Так, в статье [5] авторы рассматривали максимальный отступ (*margin*), как способ определения меры неуверенности отдельного предсказания на изображении, а также исследовали способы их агрегации по всему изображению для получения меры неуверенности всего изображения. Рассмотренные способы агрегации неуверенности:

- сумма

$$\Phi_{Sum} = \sum_{i \in D} \varphi_{1\text{vs}2}(x_i);$$

- среднее

$$\Phi_{Avg} = \frac{1}{|D|} \sum_{i \in D} \varphi_{1\text{vs}2}(x_i);$$

- максимум

$$\Phi_{Max} = \max_{i \in S} \varphi_{1\text{vs}2}(x_i).$$

Отдельно исследовался вопрос о несбалансированности классов в обучающей выборке, для чего был рассмотрен подход с применением весовой схемы.

В статье [6] рассматривались подходы активного обучения, основанные не только на работе классификатора, но и более совершенные стратегии, учитывающие локализацию объектов, а также ее устойчивость к гауссовским шумам.

1.2 Активное обучение в задаче сегментации документов

Задача сегментации документов имеет ряд важных особенностей по сравнению с сегментацией естественных изображений, что усложняет задачу и требует еще большего числа обучающих примеров. Во-первых, число объектов для обнаружения в документе превосходит число объектов для обнаружения на изображении из естественного обучающего набора (например, MS-COCO [10]). Во-вторых, имеется значительная несбалансированность предсказываемых классов; так, например, в документах можно встретить сноски, но их количество значительно меньше заголовков. Поэтому обучение с нуля модели, способной достаточно качественно сегментировать изображения документов, очень трудоемкая задача, из-за чего эта предметная область долгое время оставалась мало исследуемой.

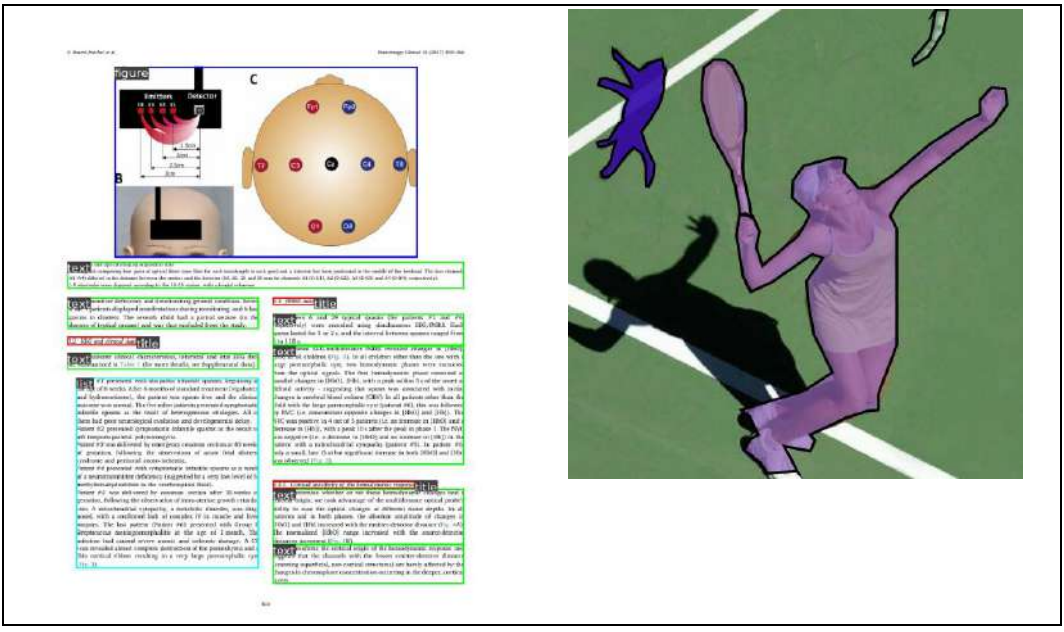


Рис. 1. Типичные примеры изображений из PubLayNet и MS-COCO
Fig. 1. Typical examples of images from PubLayNet and MS-COCO

Существенным прорывом в этой области стало появление в 2019 году обучающего набора *PubLayNet* [11]. Это крупнейший набор размеченных структурированных документов.

В 2020 году была опубликована статья [13], посвященная сегментации документов. Авторы полагают, что стандартный подход активного обучения к выбору изображений для обучения содержит существенный недостаток: методы основываются на агрегированной информативности по всему изображению, в то время как конечная цель – это определение правильной ограничивающей рамки с предсказанием для конкретного объекта на изображении, так для задач со значительной несбалансированностью классов метод может быть не оптимальным из-за чрезмерной распространенности одного из классов. Таким

образом, авторы предлагают подход, в котором оракулу нужно размечать не все изображение, а только объекты, в которых модель наименее уверена.

Однако до сих пор не существует исследований, посвященных использованию классических подходов активного обучения к задаче сегментации изображений документов. Выбирая примеры для обучения только на основании классификации, мы предположительно могли бы получать приемлемое качество, используя меньше ресурсов.

2. Общий вид алгоритма активного обучения

Введём следующие обозначения:

- множество неразмеченных объектов: $X = \{x_1, x_2, \ldots, x_n\}$;
- множество меток: $Y = \{y_1, y_2, \ldots, y_n\}$;
- хотим восстановить зависимость $a : X \to Y$.

На каждом шаге активного обучения имеем следующее:

- множество размеченных объектов: X_l;
- множество неразмеченных объектов: $X \setminus X_l$;
- множество меток для X_l: Y_l;
- функция выбора самых информативных примеров: $\nu : X \to X_i \subset X$;
- оракул, число обращений к которому минимизируется: $f : X_i \to Y_i$;
- $Xl = Xl \cup X_i, Yl = Yl \cup Y_i$.

Тогда общий вид алгоритма активного обучения может быть описан псевдокодом, показанным на Листинге 1.

```
1: инициализация весов на k произвольных изображениях:
   ((x1, y1), (x2, y2), ...,(xk, yk))
2: for i = 1 to count do
3: выбор n изображений по предсказаниям модели, в соответствии с
   выбранной стратегией:
   Φ : X \ Xl → Xn ⊂ X
4: Разметка соответствующих изображений
   f : Xn → Yn
5: обучение модели по выбранным экземплярам
6: Xl = Xl ∪ Xn, Yl = Yl ∪ Yn
7: end for
```

Листинг 1. Псевдокод модели активного обучения
Listing 1. Pseudocode for an active learning model

3. Исследуемые стратегии разметки

Так как нами не было найдено исследований классических подходов активного обучения в исследуемой предметной области, в данной работе мы исследовали следующие классические способы подсчета неуверенности отдельного предсказания:

- максимальный отступ [2],
- максимальная энтропия [3],

а также сумму и максимум в качестве функции агрегации:

- $\Phi_{Sum} = \sum_{x_i \in X} \varphi(x_i)$;
- $\Phi_{Max} = \max_{x_i \in X} \varphi(x_i)$.

Классификатор используемой модели помимо вероятностей структурных элементов документа выдаёт также вероятность фона изображения, из чего появилось предположение о том, что использование вероятности фона может быть несодержательным и искажает оценку

неуверенности, что может приводить к неоптимальному решению. Для проверки этого предположения был рассмотрен новый способ подсчета информативности изображения, не учитывающий вероятности предсказания фона изображения.

Пусть $p_1,\ p_2, \dots p_n$ – исходные вероятности, вычисленные классификатором модели, где p_1 – вероятность фона, а p_2 ,…, p_n – вероятности структурных элементов документа. Тогда переопределим вероятности следующим образом:

$$p_i^* = \frac{p_{i+1}}{\sum_{i\in\{2,\dots,n\}} p_i}, i = \overline{1, n-1}.$$

Откуда неуверенность отдельного предсказания определяется следующим образом:

$$\varphi(x)_{margin\_norm} = \left[1 - \left(\max_{c_1 \in K^*} p^*(c_1|x) - \max_{c_2 \in K^* \setminus c_1} p^*(c_2|x)\right)\right]^2,$$

$$\varphi(x)_{entropy\_norm} = -\sum_{c \in K^*} p^*(c|x) \log p(c|x),$$

где $K^* = K \setminus c_{\text{фон}}$.

4. Наборы данных

MS COCO [10] – популярный крупный (порядка 330 тысяч изображений) набор данных для задач обнаружения объектов и сегментации. *MS COCO* содержит распространенные объекты в их естественном окружении.

PubLayNet [11] – крупнейший обучающий набор, посвященный анализу структуры документов. *PubLayNet* содержит изображения статей и исследовательских работ; в нем представлены категории, приведенные в табл. 1.

Табл. 1. Список категорий в PubLayNet
Table 1. List of categories in PubLayNet

Категории согласно статье [11]	Категория
Text	Текст
Title	Заголовок
List	Список
Table	Таблица
Figure	Изображение

DLA [12] – маленький (порядка 500 документов для обучения и 300 документов для валидации) обучающий набор ручной разметки, состоящий из правовых актов на русском языке. В нем представлены категории, приведенные в табл. 2.

Табл. 2. Список категорий в DLA
Table 2. List of categories in DLA

Категории в *DLA*	Категория
Text	Текст
Table	Таблица
Figure	Изображение

5. Перенос знаний

Одним из способов сокращения обучающей выборки является перенос знаний (Transfer Learning). Суть метода заключается в том, чтобы использовать знания, полученные моделью из решения одной задачи (для которой имеется большой размеченный обучающий набор), к задаче, в которой получение большого числа обучающих примеров невозможно, либо очень затратно.

Для получения базовой модели мы выполнили перенос знаний с модели предобученной на *MS COCO* на *PubLayNet*, так как используемый фреймворк не имел модели, предобученной на документах. Далее был выполнен перенос знаний на обучающий набор *DLA*, с целью получения верхней оценки на будущее решение.

Кроме того, нас интересовало, как предметная область, на которой обучалась модель изначально, влияет на последующее дообучение. Формально, пусть в нашем распоряжении имеется 2 обучающих набора A и B с классами $\{a_1, a_2, \ldots, a_n\}$ и $\{b_1, b_2, \ldots, b_m\}$ соответственно. Даст ли модель M_B, предобученная на обучающем наборе B, преимущество при активном обучении на обучающем наборе C с классами $\{ b_1,, b_2,, \ldots, b_{p'}\}$, $p' < m$, по сравнению с моделью M_A, предобученной на обучающем наборе A?

Изначально мы попробовали применить обученную *PubLayNet*-модель для сегментации документов из *DLA*, так как она умела определять все интересующие нас классы. Однако показатель качества такой модели на новых данных был равен нулю, что свидетельствовало о том, что модели для качественного предсказания требуется дообучение даже в рамках одной предметной области. Тогда мы решили проверить, как изменится качество предсказаний выбранной модели, если исходная *MS COCO*-модель не будет предобучаться на *PubLayNet*, а сразу будет активно обучаться на небольшом наборе *DLA* из другой предметной области.

6. Эксперименты

6.1 Показатель качества

Популярный и часто используемый показатель качества в задаче сегментации – модифицированный *Mean Average Precision*. Это обычный *mAP*, усредненный для 10 пороговых значений *IoU* с шагом 0.05: [*IoU*=0.50:0.95]. Авторы метрики [10] заявляют, что усреднение по *IoU* позволяет лучше локализовать объекты.

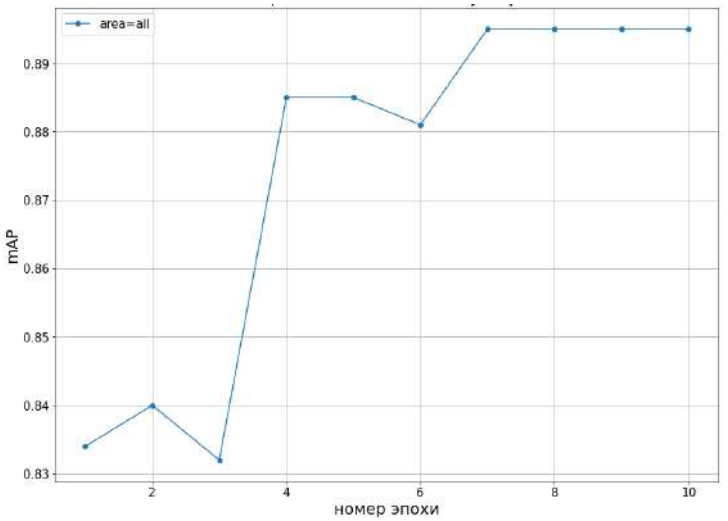


Рис. 2. Перенос знаний с MS COCO на PubLayNet
Fig. 2. Transfer of knowledge from MS COCO to PubLayNet

6.2 Модель

В силу структуры документов задачи семантической сегментации, сегментации экземпляров и обнаружения объектов являются практически эквивалентными в этой предметной области. В частности, это подтверждают результаты, полученные авторами статьи [11], они обучали

модель сегментации экземпляров (*Mask RCNN* [15]), а также модель обнаружения объектов (*Faster RCNN* [14]). Существенных отличий в качестве предсказаний моделей не было, а так как наш метод для определения степени неуверенности изображения использует только результаты работы классификатора, то использование более тяжелой модели мы посчитали неразумным, поэтому нами была выбрана модель *Faster RCNN*. Плюсом является то, что в *PyTorch*[1] имеется реализация *Faster RCNN*, предобученная на *MS COCO*.

Так как у нас не было модели *Faster RCNN*, предобученной на *PubLayNet*, а обучение было очень затратным, мы выполнили перенос знаний с *MS COCO* на 1/7 часть *PubLayNet* с результатами на отложенной выборке *PubLayNet*-а, показанными на рис. 2.

Полученная модель имитировала модель, предобученную на *PubLayNet*.

6.3 Дизайн экспериментов

6.3.1 Выбор стратегии сэмплинга

Модель, предобученная на *PubLayNet*, инициализировала веса выходного слоя на 10 произвольных изображениях из *DLA* в течение 10 эпох обучения. Эта модель сохранялась и была начальной для всех последующих экспериментов с различными стратегиями активного обучения. Далее для каждой стратегии выбора экземпляров по степени неуверенности следовало 49 итераций активного обучения, на каждой итерации модель получала 8 новых примеров с наибольшей мерой неуверенности и дообучалась на всех размеченных на текущий момент данных($8 \times$ номер итерации) в течение 10 эпох.

Для получения более объективных результатов был реализован скользящий контроль с разбиением на 5 групп. После инициализации выходного слоя, в нашем распоряжении было 490 изображений для обучения, то есть одна группа состояла из 98 изображений. Таким образом, на первой итерации обучение происходило на 8 изображениях, а на 49-й итерации – на $490 \times \frac{1}{5} \times\ 4 = 392$ изображениях; валидация все время осуществлялась на оставшихся 98 изображениях(как это и устроено в классическом скользящем контроле).

6.3.2 Критерий выбора стратегии

Получение показателя качества, соответствующего верхней границе, на меньшем числе обучающих примеров – малореальная задача. Тем не менее, нам нужно формализовать то, что мы будем считать достаточно качественным обучением в условиях минимизации обращений к оракулу для разметки.

Мы предполагаем, что на ранних итерациях активный выбор обучающих примеров приведет к более быстрому обучению (по сравнению с произвольным выбором примеров для обучения) до некоторого порогового значения, которое впоследствии будет не сильно меняться при добавлении новых размеченных данных. Под достаточно качественной моделью будем понимать такую, что ее относительное качество не будет меняться на 1 процент и более в течение трех итераций дообучения на новых данных. Таким образом, мы получим модель, показатель качества которой относительно быстро выходит на некоторое плато, а затем мало изменяется при добавлении новых обучающих примеров.

6.4 Результаты

6.4.1 Выбор стратегии сэмплинга

На рис. 3 приведен график зависимости среднего показателя качества модели в зависимости от числа итераций активного обучения (фактически, числа обучающих примеров). Из-за

[1] https://pytorch.org/

количества рассмотренных стратегий выбора примеров для обучения, а также схожести многих из них на некоторых участках, по данному графику трудно делать выводы о качестве какой-то конкретной модели, тем не менее можно отметить, что на начальных этапах обучения (≤ 20 итераций) *random* и *margin_max* отстают от остальных стратегий, а к концу обучения(≥ 30 итераций) все стратегии выходят на плато.

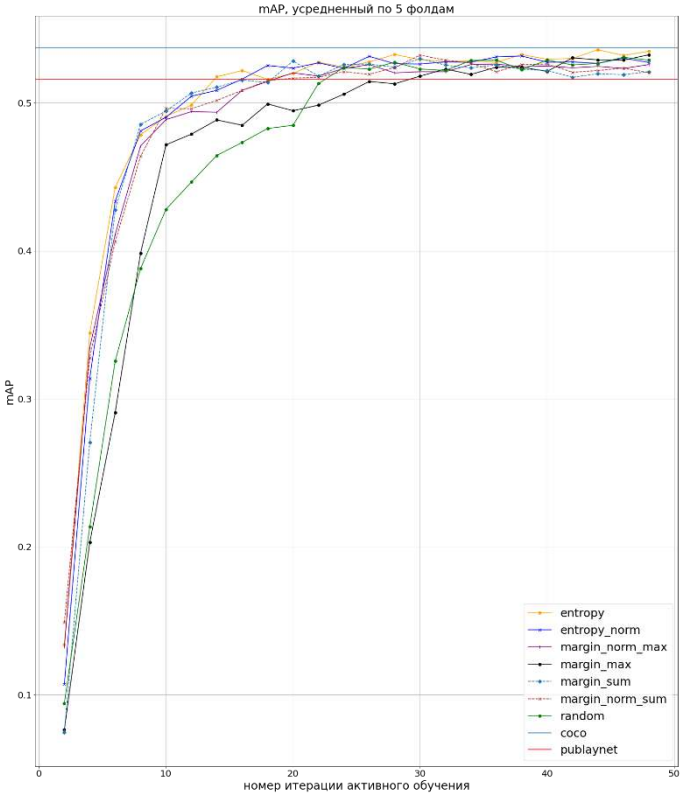


Рис. 3. Усредненный по 5-ти фолдам mAP
Fig. 3. mAP averaged over 5 folds

Также на рис. 3 приведен результат моделей, для которых была выполнена перекрестная проверка переноса знаний с *MC COCO* и *PubLayNet*.

Для сравнения разных стратегий активного обучения приводим диаграммы размаха (рис. 4) показателя качества для разных стратегий активного обучения в зависимости от номера итерации. Также на графиках отображены средние значения показателя качества модели, выбиравшей примеры для обучения произвольно.

margin_sum, entropy, entropy_norm имеют наибольший прирост по сравнению с *random* стратегией; кроме того, их отличает наименьшая величина межквартильного размаха, что говорит о небольшом разбросе рассматриваемых значений, то есть качество модели не сильно зависит от того, на каких данных обучались модели.

Худший результат демонстрирует *margin_max* с минимальными значениями медианы, а также наибольшим значением межквартильного размаха. При этом использование нормировки позволяет быстрее выйти на плато в области mAP ≥ 0.5, а также уменьшить разброс значений показателя качества модели.

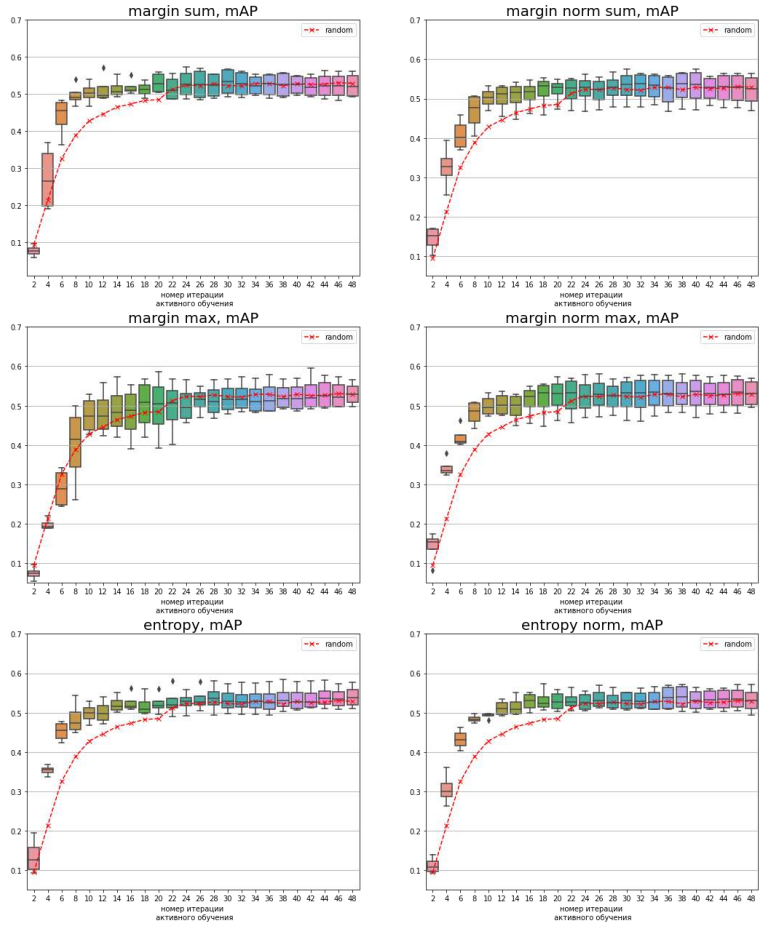


Рис. 4. Сравнение разных стратегий активного обучения
Fig. 4. Comparison of different active learning strategies

В соответствии с критерием отбора (п. 6.3.2) приведем также табл. 3, демонстрирующую, на какой итерации разные стратегии стали удовлетворять критерию.

Табл. 3. Сравнение разных стратегий активного обучения
Table 3. Comparison of different active learning strategies

Стратегия	**№ итерации**	$\overline{mAP_{CV}}$	**Transfer$_1$**	**Transfer$_2$**
margin sum	12	0.507	0.98	0.94
entropy	14	0.518	1.00	0.96
entropy norm	18	0.525	1.01	0.98
margin norm sum	18	0.515	0.99	0.96
random	24	0.524	1.02	0.98
margin norm max	24	0.523	0.99	0.97
margin max	30	0.518	1.00	0.96
$Transfer_{pyblaynet}$	-	0.516	1.00	0.96
$Transfer_{coco}$	-	0.537	1.04	1.00

Как видно из табл. 3, стратегии выходят на плато добиваясь 98%-го (и более) показателя качества от переноса знаний с *PubLayNet* и 94%-го (и более) показателя качества от переноса знаний с *MS COCO*. Кроме того, перенос знаний с *PubLayNet* показал худшее качество по сравнению с переносом знаний с *MS COCO*, несмотря на то, что классы обучающего набора, на котором проходили эксперименты, содержатся в *PubLayNet* и отсутствуют в *MS COCO*.

Для полноты картины приведем также табл. 4, которая показывает, какого показателя качества удалось добиться каждой модели к итерации, когда некоторая стратегия вышла на плато согласно критерию.

Табл. 4. Зависимость mAP_{CV} от номера итерации
Table 4. Dependence of mAP_{CV} on the iteration number

Стратегия/итерация	12	14	18	24	30
margin sum	**0.507**	0.511	0.514	0.526	0.529
entropy	0.498	**0.518**	0.516	0.524	0.529
entropy norm	0.504	0.508	**0.525**	0.523	0.526
margin norm sum	0.496	0.502	**0.515**	0.521	0.532
random	0.446	0.464	0.482	**0.524**	0.523
margin norm max	0.494	0.4936	0.514	**0.523**	0.521
margin max	0.479	0.488	0.499	0.506	**0.518**

Таким образом, использование в качестве меры неуверенности *margin* (или *entropy*) позволяет к 12-й (14-й) итерации получить 94% (96%) качества от переноса знаний с *MS COCO*, используя лишь 10 + 12×8 = 106 (10 + 14×8 = 122) изображений из 392. В то же время произвольный выбор изображений для обучения показывает лишь 83% (86%) качества модели, обучающейся на всех данных.

6.4.2 Перенос знаний

Также были проведены дополнительные эксперименты по исследованию зависимости качества активного обучения от инициализации начальной модели.

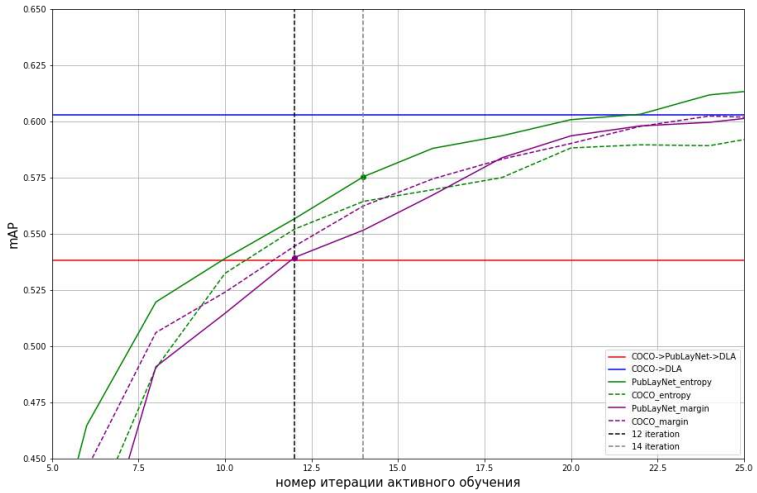


Рис. 5. Качество активного обучения на отложенной выборке в зависимости от инициализации
Fig. 5. The quality of active learning on deferred sampling depending on initialization

На рис. 5 приведены зависимости двух лучших стратегии активного обучения из предыдущего пункта для разных инициализаций на отложенной выборке: инициализация моделями, предобученными на *PubLayNet* и *MS COCO*.

Лучшее качество демонстрирует модель, предобученная на *PubLayNet* с использованием *entropy* в качестве меры неуверенности. Однако сказать, что модель, предобученная на смежном обучающем наборе, даёт преимущество при активном обучении, нельзя, так как *margin* с инициализацией на *MS COCO* превосходит аналогичную модель с инициализацией на *PubLayNet*.

7. Заключение

Использование классических подходов активного обучения в задаче сегментации изображений документов можно считать эффективным, так они позволяют получать высокое качество предсказания, используя лишь около четверти данных, требуемых для переноса знаний.

Модифицированные подходы активного обучения на основе перенормировки исходных вероятностей не показали преимущества по сравнению с классическими стратегиями выбора по степени неуверенности.

Перенос знаний на смежный обучающий набор не показал преимущества по сравнению с переносом знаний с обучающего набора с произвольными классами.

Список литературы / References

[1] Settles B. Active learning literature survey. Technical Report #1648, University of Wisconsin-Madison, Department of Computer Sciences, 2009, 47 p.
[2] Scheffer T., Decomain C., Wrobel S. Active hidden markov models for information extraction. In Proc. of the International Symposium on Intelligent Data Analysis, 2001, pp. 309-318.
[3] Dagan I., Engelson S. Committee-based sampling for training probabilistic classifiers. In Proc. of the Twelfth International Conference on Machine Learning, 1995, pp. 150-157.
[4] Culotta A., McCallum A. Reducing labeling effort for structured prediction tasks. In Proc. of the 20th National Conference on Artificial Intelligence, 2005, pp. 746-751.
[5] Brust C., Käding C., Denzler J. Active Learning for Deep Object Detection. In Proc. of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, 2019, pp. 181-190.
[6] Kao C., Lee T. et al. Localization-Aware Active Learning for Object Detection. In Proc. of the 14th Asian Conference on Computer Vision, 2018, pp. 506-522.
[7] Roy S., Unmesh A., Namboodiri V. Deep active learning for object detection. In Proc. of the 29th British Machine Vision Conference, 2018, 12 p.
[8] Aghdam H., Gonzalez-Garcia A. et al. Active Learning for Deep Detection Neural Networks. In Proc. of the 17th IEEE/CVF International Conference On Computer Vision, 2019, pp. 3671-3679.
[9] Lv X., Duan F. et al. Deep active learning for surface defect detection. Sensors, vol. 20, no. 6, 2020, article no. 1650.
[10] Lin T., Maire M. et al. Microsoft COCO: Common Objects in Context. Lecture Notes in Computer Science, vol. 8693, 2014, pp. 740-755.
[11] Zhong X., Tang J., Yepes A. PubLayNet: largest dataset ever for document layout analysis. In Proc. of the International Conference on Document Analysis and Recognition (ICDAR), 2019, pp. 1015-1022.
[12] Беляева О.В., Перминов А.И., Козлов И.С. Использование синтетических данных для тонкой настройки моделей сегментации документов. Труды ИСП РАН, том 32, вып. 4, 2020 г., стр. 189-202 / Belyaeva O.V., Perminov A.I., Kozlov I.S. Synthetic data usage for document segmentation models fine-tuning. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 4, 2020. pp. 189-202 (in Russian). DOI: 10.15514/ISPRAS–2020–32(4)–14
[13] Shen Z., Zhao J. et al. OLALA: Object-Level Active Learning for Efficient Document Layout Annotation. arXiv:2010.01762, 2021, 12 p.
[14] Ren S., He K. et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In Proc. of the 28th International Conference on Neural Information Processing Systems, 2015, pp. 91-99.
[15] He K., Gkioxari G.et al. Mask R-CNN. In Proc. of the IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2980-2988.

Информация об авторах / Information about authors

Дмитрий Маратович КИРАНОВ – студент магистратуры МФТИ, лаборант в ИСП РАН. Научные интересы: активное обучение, непрерывное обучение.

Dmitry Maratovich KIRANOV – MIPT master’s student, laboratory assistant at ISP RAS. Research interests: active learning, lifelong learning.

Максим Алексеевич РЫНДИН – аспирант. Научные интересы: методы адаптации к домену и переноса знаний, онлайн обучение, обработка текстов на естественном языке, векторное представление слов/предложений/текстов, генеративные модели, активное и проактивное обучение, анализ социальных сетей.

Maxim Alexeevitch RYNDIN – PhD Student. Research interests: domain adaptation, transfer learning, online learning, natural language processing, embeddings, generative models, active and proactive learning, social media analysis.

Илья Сергеевич КОЗЛОВ является стажером-исследователем. Научные интересы: распознавание структуры документов, цифровая обработка изображений, нейросетевая обработка данных, распознавание образов.

Ilya Sergeevich KOZLOV – researcher at ISP RAN. Research interests: document layout analysis, digital image processing, neural network data processing, image pattern recognition.

DOI: 10.15514/ISPRAS-2021-33(6)-15

Межъязыковой перенос знаний при извлечении информации о лекарствах из пользовательских текстов

[1, 2] *А.С. Саховский, ORCID: 0000-0003-2762-2910 <andrey.sakhovskiy@gmail.com>*
[2, 3, 4] *Е.В. Тутубалина, ORCID: 0000-0001-7936-0284 <tutubalinaev@gmail.com>*

[1] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1.
[2] *Казанский федеральный университет,*
420008, Россия, Казань, ул. Кремлевская, д.18.
[3] *Национальный исследовательский университет "Высшая школа экономики",*
101000, Россия, Москва, ул. Мясницкая, д.20
[4] *Sber AI*
121170, Россия, Москва, Кутузовский пр-кт, д. 32

Аннотация. Задача извлечения именованных сущностей, соответствующих лекарствам, заболеваниям и лекарственным реакциям, из текстов различных предметных областей и языков является основополагающим компонентом многих приложений, основанных на извлечении информации из текстов. В данной работе производится оценка эффективности многоязыковых моделей, основанных на архитектуре BERT, для решения задач распознавания именованных сущностей медицинской направленности и многоклассовой классификации предложений. В ходе экспериментов было исследовано влияние переноса знаний между двумя англоязычными корпусами и одним русскоязычным корпусом размеченных отзывов о лекарственных препаратах. Рассмотренные корпуса содержат разметку на уровне предложений, обозначающую присутствие или отсутствие в них медицинских сущностей некоторого типа. Предложения, принадлежащие некоторому классу, содержат дополнительную разметку на уровне сущностей, позволяющую установить принадлежность отдельных выражений к сущностям некоторого типа, таким, как название, показание к применению или эффект лекарства. Результаты экспериментов показали, что для русского языка наибольшая эффективность переноса знаний при предобучении моделей BERT на коллекции, состоящей из 5 миллионов неразмеченных русскоязычных и англоязычных пользовательских отзывах, наблюдается при распознавании побочных эффектов лекарств. Для задачи распознавания именованных сущностей наилучшее значение макро F-меры, равное 74,85%, показала модель RuDR-BERT, предобученная на русскоязычных текстах медицинской предметной области. Для задачи классификации наилучшее значение макро F-меры, равное 70%, показала модель EnRuDR-BERT, предобученная на русскоязычных и англоязычных текстах медицинской направленности. Превосходство данной модели над моделью BERT, предобученной на текстах общей предметной области, составляет 8,64% макро F-меры.

Ключевые слова: обработка естественного языка; классификация текстов; извлечение информации; распознавание именованных сущностей; BERT

Для цитирования: Саховский А.С., Тутубалина Е.В. Межъязыковой перенос знаний при извлечении информации о лекарствах из пользовательских текстов. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 217-228. DOI: 10.15514/ISPRAS–2021–33(6)–15

Благодарности: Данная работа выполнена при поддержке гранта Президента РФ МК-3193.2021.1.6

Cross-lingual transfer learning in drug-related information extraction from user-generated texts

[1, 2] *A.S. Sakhovskiy ORCID: 0000-0003-2762-2910 <andrey.sakhovskiy@gmail.com>*
[2, 3, 4] *E.V. Tutubalina ORCID: 0000-0001-7936-0284 <tutubalinaev@gmail.com>*

[1] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia.
[2] *Kazan Federal University,*
18 Kremlyovskaya street, Kazan, 420008, Russia.
[3] *National research university Higher school of economics,*
20 Myasnitskaya street, Moscow, 101000, Russia.
[4] *Sber AI,*
32 Kutuzovskiy prospect, Moscow, 121170, Russia

Abstract. Aggregating knowledge about drug, disease, and drug reaction entities across a broader range of domains and languages is critical for information extraction (IE) applications. In this work, we present a fine-grained evaluation intended to understand the efficiency of multilingual BERT-based models for biomedical named entity recognition (NER) and multi-label sentence classification tasks. We investigate the role of transfer learning (TL) strategies between two English corpora and a novel annotated corpus of Russian reviews about drug therapy. Labels for sentences include health-related issues or their absence. The sentences with one are additionally labelled at the expression level to identify fine-grained subtypes such as drug names, drug indications, and drug reactions. Evaluation results demonstrate that BERT trained on Russian and English raw reviews (5M in total) shows the best transfer capabilities on evaluation of adverse drug reactions on Russian data. The macro F1 score of 74.85% in the NER task was achieved by our RuDR-BERT model. For the classification task, our EnRuDR-BERT model achieves the macro F1 score of 70%, gaining 8.64% over the score of a general domain BERT model.

Keywords: natural language processing; text classification; information extraction; named entity recognition; BERT.

For citation: Sakhovskiy A.S., Tutubalina E.V. Cross-lingual transfer learning in drug-related information extraction from user-generated texts. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 217-228 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–15

Acknowledgements. The work has been supported by a grant from the President of the Russian Federation for young scientists-candidates of science (MK-3193.2021.1.6).

1. Введение

Значительная часть существующих работ в области обработки текстов биомедицинской тематики посвящена обработке англоязычных текстов. В частности, обработке англоязычных научных текстов, например, научных аннотаций. В работе [1] произведен обзор данной предметной области для английского языка. Однако задача обработки пользовательских текстов биомедицинской тематики для языков, отличных от английского, в настоящий момент слабо изучена. Значительные продвижения в области разработки многоязыковых нейросетевых моделей обработки текстов, в частности, предобученных языковых моделей, основанных на архитектуре Transformer [2], позволяют получать информативные векторные представления слов, зависящие от окружающего словесного контекста [3, 4, 5, 6]. Данные продвижения позволяют выдвинуть предположение о возможности улучшения качества и разработки более эффективных моделей для широкого круга задач обработки естественного языка, включая классификацию текстов и извлечение именованных сущностей в биомедицинской предметной области.

Одним из значимых направлений современных исследований в области обработки естественного языка является разработка методов так называемого переноса знаний, суть которого состоит в использовании информации, полученной некоторой моделью при

решении одной задачи, для решения некоторой другой. Одним из проявлений переноса знаний может являться использование предобученных языковых моделей. Идея предобучения языковых моделей состоит в обучении некоторой модели на огромном неразмеченном наборе данных с целью извлечения из данного набора данных информации, которая в дальнейшем может быть использована для решения другой задачи. Результатом предобучения является некоторая предобученная модель, позволяющая получать информативные векторные представления текста и его отдельных слов.

Ввиду неравномерного развития ресурсной базы для различных языков, особую значимость приобретает задача межъязыкового переноса знаний, состоящая в использовании знаний и ресурсов, разработанных для одного языка, при решении задач на некотором другом языке.

В отличие от предшествующих исследований, рассматривавших перенос знаний в рамках текстов одного языка, ключевым аспектом данной работы является изучение эффективности межъязыкового переноса знаний в области биомедицинских текстов. В рамках данной работы была проведена оценка эффективности (i) различных предобученных моделей архитектуры BERT при переносе знаний внутри русского языка, а также (ii) методов межъязыкового переноса знаний из английского языка в русский применительно к текстам медицинской направленности. В качестве задач были рассмотрены (i) задача классификации предложений и (ii) извлечение именованных сущностей применительно к пользовательским текстам медицинской тематики. Целью данного исследования является оценка возможности использования английского языка, для которого существует развитая ресурсная база, для улучшения качества извлечения информации о лекарствах и заболеваниях из текстов на русском языке. Для проведения экспериментов по оценке данной возможности в данной работе используется русскоязычный корпус Russian Drug Reaction Corpus (RuDReC) [7], предназначенный для исследований в области извлечения из текстов информации о лекарствах и их побочных эффектах, а также о заболеваниях. Данная работа продолжает исследование эффективности переноса знаний для русского языка, начатое в работе [7].

2. Обзор предметной области

В последние годы значительная часть работ, посвященных задачам извлечения информации из текстов о лекарствах, основана на использовании англоязычных текстов социальных сетей, пользовательских отзывов и медицинских записей [8, 9, 10, 11]. Однако существует небольшое число работ, посвященных другим языкам. Так, в работе [12] были рассмотрены описания лекарств на испанском языке, а в работах [13, 14] были использованы сертификаты о смерти на французском языке. Что касается русского языка, в работе [15] были рассмотрены клинические записи на русском языке, а в работах [16, 17] – русскоязычные записи пользователей социальной сети Twitter.

В отличие от английского языка, число доступных размеченных корпусов, посвященных извлечению информации о лекарствах и заболеваниях, для русского языка относительно мало. Среди существующих корпусов можно выделить следующие. В работе [18] был представлен корпус русскоязычных отзывов о лекарственных препаратах, содержащий разметку на уровне предложений. Корпус предоставляет разметку по 4 классам в зависимости об информации, содержащейся в предложении, включая побочные эффекты, показания к применению и информацию об эффективности лекарственного препарата. Отдельно необходимо выделить русскоязычные корпуса соревнований Social Media Mining for Health Applications (SMM4H) 2020 и 2021 годов [19, 20]. В рамках данных соревнований были представлены корпуса русскоязычных твитов, размеченных на предмет содержания в них упоминаний побочных эффектов.

3. Данные

В данной работе было проведено исследование эффективности предобученных моделей архитектуры BERT для решения задач классификации текстов и извлечения именованных сущностей медицинской тематики на русском языке. Для обучения и оценки качества моделей был использован корпус RuDReC. The Russian Drug Reaction Corpus [7] (RuDReC) – корпус русскоязычных пользовательских отзывов о лекарственных препаратах. Корпус доступен для исследовательских целей по ссылке: https://github.com/cimm-kzn/RuDReC. Данный корпус состоит из двух частей: (i) неразмеченной части, состоящей из 1,4 миллионов текстов и (ii) размеченной части, включающей 500 текстов. Отзывы размеченной части корпуса содержат разметку как на уровне предложений, так и на уровне сущности. Каждое предложение может принадлежать одному или нескольким классам предложений из следующего списка:

- предложение, в котором сообщается о положительном эффекте приема лекарственного препарата (класс DE);
- предложение, в котором сообщается о том, что в результате приема лекарственного препарата состояние пациента осталось неизменным или ухудшилось (класс DIE);
- предложение, в котором содержится упоминание симптомов и показаний к применению, побудивших пользователя к приему лекарства (класс DI);
- предложение, в котором содержится упоминание нежелательных побочных эффектов, возникших в результате приема лекарства (класс ADR);
- предложение, содержащее упоминание некоторого события или эффекта, связанного с болезнью или лекарственным препаратов (класс Finding). Предложения данного типа могут содержать историю болезни пациента, название лекарства, сообщать об отсутствии ожидаемого побочного эффекта.

Статистика по числу размеченных предложений каждого класса приведена в табл. 1. Как можно заметить, наибольшее число предложений представлено для класса симптомов (DI), а наименьшим числом примеров обладает класс Finding.

Табл. 1. Статистика по числу размеченных предложений различных классов корпуса RuDReC
Table 1. RuDReC corpus statistics on number of sentences of different classes

Общее число предложений	**DE**	**DIE**	**ADR**	**DI**	**Finding**
4855	424	278	379	949	172

Помимо разметки на уровне предложений тексты корпуса RuDReC содержат разметку на уровне сущностей, что позволяет использовать данный корпус для обучения и оценки моделей извлечения именованных сущностей медицинской тематики. Размеченные сущности принадлежат одному из следующих 6 возможных типов: (i) ADR, (ii) DI, (iii) Finding, (iv) Drugclass (класс лекарственного препарата. Например, противовирусный или противовоспалительный препарат и др.), (v) Drugform (лекарственная форма препарата. Например, таблетки, микстура, мазь и др.), (vi) Drugname (название лекарства или его действующего вещества).

Помимо русскоязычного корпуса RuDReC, в данной работе были использованы англоязычные корпуса PsyTAR [9] и CADEC [10]. Psychiatric Treatment Adverse Reactions (PsyTAR) – корпус, состоящий из 887 размеченных пользовательских отзывов о психиатрических препаратах. По структуре и содержанию разметки данный корпус имеет высокое сходство с размеченной частью корпуса RuDReC. Так, данный корпус содержит разметку как на уровне предложений, так и на уровне сущностей. Согласно разметке на уровне предложений, каждое предложение может быть отнесено к одному или нескольким из 7 классов. Набор возможных классов содержит классы, аналогичные классам DE, DIE,

ADR, DI, Finding. Основное отличие в наборе классов корпуса PsyTAR от набора корпуса RuDReC состоит в наличии двух дополнительных классов предложений: (i) сообщающих о том, что прекращение использования лекарства привело к возникновению у пользователя синдрома отмены (класс WD); (ii) сообщающих о некотором испытанном пользователем симптоме, не являющемся ни причиной применения, ни побочным эффектом приема лекарства (класс SSI). В рамках данной работы при обучении моделей на корпусе PsyTAR была использована разметка только по тем классам предложений, для которых существует аналогичный класс в корпусе RuDReC, т.е. классы WD и SSI не рассматривались. Разметка на уровне сущностей содержит включает сущности 4 типов: ADR, WD, DI, SSI.

В табл. 2 представлена статистика по числу размеченных предложений корпуса PsyTAR. Приведенная статистика позволяет сделать следующие основные наблюдения. Во-первых, наиболее частотным классом предложений является класс предложений, содержащих упоминание побочных эффектов. Число предложений данного класса в корпусе PsyTAR более, чем в 5 раз выше, чем в корпусе RuDReC (2168 и 379 для корпусов PsyTAR и RuDReC соответственно). Во-вторых, число предложений класса Finding более, чем в 10 больше в корпусе PsyTAR, чем в корпусе RuDReC (2107 и 172 соответственно).

Табл. 2. Статистика по числу размеченных предложений различных классов корпуса PsyTAR
Table 2. PsyTAR corpus statistics on number of sentences of different classes

Общее число предложений	**DE**	**DIE**	**ADR**	**DI**	**Finding**
6004	1087	337	2168	517	2107

Корпус CSIRO Adverse Drug Event Corpus (CADEC) состоит из 1253 пользовательских отзывов о 12 лекарственных препаратах. Данный корпус предоставляет разметку сущностей 5 типов: (i) название лекарства, (ii) ADR, (iii) заболевание и (iv) его симптомы, побудившие пользователя к приему лекарства, (v) Finding. В данной работе мы рассматриваем сущности классов (iii) и (iv) как сущности одного общего класса DI.

4. Модели

В данной работе были использованы следующие предобученные модели архитектуры Bidirectional Encoder Representations from Transformers (BERT) [3]:

1) Multi-BERT[1] – многоязыковая модель BERT, предобученная на текстах Википедии на 104 языках.
2) RuBERT [21] – многоязыковая модель BERT, дополнительно предобученная на русскоязычных текстах Википедии и новостных текстах. Для инициализации данной модели была использована модель Multi-BERT с измененным словарем
3) RuDR-BERT[2] [7] – многоязыковая модель BERT, дополнительно предобученная на неразмеченной части корпуса RuDReC [7], содержащей 1,4 миллионов отзывов о лекарственных препаратах. В качестве инициализации для данной модели была использована модель Multi-BERT;
4) EnRuDR-BERT[3] – многоязыковая модель BERT, предобученная на неразмеченной части корпуса RuDReC и англоязычном корпусе пользовательских текстов о лекарствах [22]. Объем англоязычного корпуса составляет около 2,6 миллионов текстов.

Каждая из использованных моделей содержит 12 слоев, 12 голов интерактивного внимания и имеет размерность векторных представлений, равную 768. В рамках данной работы для

[1] https://github.com/google-research/bert
[2] https://huggingface.co/cimm-kzn/rudr-bert
[3] https://huggingface.co/cimm-kzn/enrudr-bert

проведения экспериментов с предобученными моделями архитектуры BERT была использована программная реализация оригинальной модели BERT, доступная по адресу: https://github.com/google-research/bert.

5. Эксперименты

5.1. Классификация предложений

В данной работе для задачи классификации предложений было исследовано влияние переноса знаний на качество решения задачи на русском языке. В ходе экспериментов было исследовано влияние следующих способов переноса знаний: (i) перенос знаний в рамках одного языка путем предобучения языковых моделей BERT на текстах целевого языка и целевой предметной области; (ii) перенос знаний путем последовательного обучения модели на текстах вспомогательного и целевого языка на схожей задаче; (iii) перенос знаний путем обучения модели на вспомогательном языке и оценке качества классификации на целевом языке без обучения на текстах целевого языка (zero-shot перенос). Подход (iii) позволяет решать задачу на языках, для которых не существует размеченной тренировочной выборки за счет использования размеченной выборки на другом языке. В данной работе в качестве вспомогательного языка рассматривается английский, а в качестве целевого – русский язык.

В экспериментах, посвященных оценке влияния предобучения на русском языке, было произведено сравнение моделей Multi-BERT, RuBERT, RuDR-BERT, EnRuDR-BERT. Для обучения моделей был использован русскоязычный корпус RuDReC. При оценке влияния последовательного обучения сначала на английских, а затем на русских данных, были использованы англоязычный корпус PsyTAR и корпус RuDReC соответственно. Данные корпуса обладают схожими структурой разметки и набором возможных классов предложений, что позволяет переиспользовать одну и ту же модель для обучения на обоих корпусах без внесения изменений в ее архитектуру. В рамках данной серии экспериментов были рассмотрены русскоязычная модель RuDR-BERT и англо-русская модель EnRuDR-BERT. Эксперименты по оценке качества классификации при обучении на вспомогательном (английском) языке без обучения на целевом (русском) состояли в обучении моделей только на англоязычных текстах корпуса PsyTAR и оценке на корпусе RuDReC без дополнительного обучения на русскоязычных текстах. В рамках данных экспериментов также были рассмотрены модели RuDR-BERT и EnRuDR-BERT.

Для оценки эффективности каждого из рассмотренных подходов была использована процедура скользящего контроля с числом разбиений, равным 5. Каждый классификатор состоит из некоторой модели BERT и полносвязной нейронной сети. Полносвязная нейронная сеть принимает на вход векторное представление специального токена начала предложения. Обучение каждой модели происходило в течение 10 тренировочных эпох с кросс-энтропийной функцией потерь. Для сравнения качества различных обученных моделей классификации были использованы размеченные предложения русскоязычного корпуса RuDReC. В качестве метрики оценки качества моделей использована F-мера. Результаты соответствующих экспериментов представлены в табл. 3.

На основе полученных результатов можно сделать следующие основные наблюдения. Во-первых, многоязыковая модель EnRuDR-BERT значительно превзошла RuDR-BERT по макро F-мере (+7,3%) при обучении только на англоязычных данных. Наибольший прирост качества наблюдается для предложений, содержащих упоминания побочных эффектов (+28,6%) и предложений, сообщающих о положительном эффекте лекарства (+9,8%). Для предложений класса Finding превосходство модели EnRuDR-BERT незначительно, а для предложений, сообщающих о неэффективности лекарства и для предложений, содержащих упоминания симптомов, данная модель и вовсе уступила модели RuDR-BERT на 2,24% и 0,73% F-меры соответственно.

Табл. 3. Оценки F-меры предобученных моделей BERT на задаче классификации предложений корпуса RuDReC

Table 3. Performance of pretrained BERT models on the classification of RuDReC corpus sentences in terms of F1-score

Модель	**DE**	**DIE**	**ADR**	**DI**	**Finding**	**Макро F-мера**
Модели, обученные только на англоязычном корпусе PsyTAR (zero-shot перенос)						
RuDR-BERT, PsyTAR	41,69	59,91	36,29	18,05	2,22	31,63
EnRuDR-BERT, PsyTAR	51,51	57,67	64,93	17,32	3,15	38,92
Модели, обученные только на корпусе RuDReC						
RuBERT	67,7	62,27	66,65	81,63	28,51	61,35
Multi-BERT	63,61	60,19	63,45	79,58	24,32	58,23
RuDR-BERT	76,61	72,06	74,15	85,06	36,24	68,82
EnRuDR-BERT	78,01	74,47	**75,54**	85,37	35,47	69,77
Модели, последовательно обученные на английских и русских данных						
RuDR-BERT, CADEC+RuDReC	77,72	**75,78**	74,14	85,69	33,86	69,44
RuDR-BERT, PsyTAR+RuDReC	77,87	73,5	74,32	85,5	30,7	68,38
EnRuDR-BERT, PsyTAR+RuDReC	77,68	71,99	75,43	85,62	**39,3**	**70,0**
EnRuDR-BERT, CADEC+RuDReC	**78,58**	72,19	75,51	**86,31**	36,71	69,86

Во-вторых, при обучении и оценке на русскоязычных данных наивысшие значения F-меры демонстрируют модели EnRuDR-BERT и RuDR-BERT, предобученные на русскоязычных текстах целевой медицинской предметной области. В частности, модель EnRuDR-BERT, предобученная на англо- и русскоязычных текстах, превзошла русскоязычную модель RuBERT на 8,4% макро F-меры, а наибольший прирост F-меры наблюдается для классов DE (+10,3%), DIE (+12,2%), ADR (+8,9%). Наихудшие значения F-меры для всех классов показала многоязыковая модель Multi-BERT, которая, в отличие от остальных рассмотренных моделей, не проходила дополнительное предобучение на русскоязычных текстах. Данное наблюдение свидетельствует об эффективности предобучения на текстах целевого (русского) языка. Однако на эффективность предобучения существенно влияет предметная область текстов, на которых происходит предобучение. Так, несмотря на превосходство модели RuBERT, предобученной на русскоязычных новостных текстах, над моделью Multi-BERT, данная модель уступает моделям RuDR-BERT и EnRuDR-BERT, прошедшим предобучение на текстах медицинской тематики, то есть на текстах той же предметной области, что и тексты целевой задачи классификации.

В-третьих, последовательное обучение на англоязычных и русскоязычных данных не привело к существенному улучшению качества классификации с точки зрения макро F-меры, однако последовательное обучение модели EnRuDR-BERT на корпусах PsyTAR и RuDReC

привело к увеличению F-меры класса Finding на 3,8% по сравнению с обучением только на корпусе RuDReC. Наконец, для всех проведенных экспериментов наихудшие оценки качества наблюдаются для класса Finding, что может объясняться малым количеством тренировочных примеров данного класса.

5.2. Распознавание именованных сущностей

По аналогии с задачей классификации предложений, в данной работе было исследовано влияние предобучения на качество решения решения задачи распознавания именованных сущностей медицинской тематики. В ходе экспериментов было произведено сравнение следующих моделей: (i) многоязыковой модели Multi-BERT; многоязыковых моделей (ii) RuBERT и (iii) RuDR-BERT, прошедших предобучение на русскоязычных текстах; (iv) многоязыковой модели EnRuDR-BERT. Кроме того, была проведена оценка эффективности переноса знаний из английского языка путем последовательного обучения сначала на вспомогательном англоязычном корпусе с последующим дообучением на корпусе RuDReC.

Для оценки качества моделей была использована процедура скользящего контроля с 5 разбиениями. Архитектура классификатора состоит из предобученной модели BERT с дополнительным слоем softmax. Обучение каждой модели происходило в течение 40 тренировочных эпох. Результаты соответствующих экспериментов представлены в таблице 4.

Полученные результаты позволяют сделать следующие основные наблюдения. Во-первых, как и в случае задачи классификации предложений модели RuDR-BERT и EnRuDR-BERT, предобученные на текстах медицинской предметной области, показали наилучшие значения макро F-меры. Данное наблюдение позволяет сделать вывод об эффективности предобучения на текстах предметной области и языка целевой задачи. Во-вторых, аналогично задаче классификации, качество распознавания сущностей типа Finding значительно ниже, чем сущностей побочных эффектов и симптомов. В-третьих, качество распознавания сущностей, связанных с лекарствами – их названиями, классами и лекарственными формами – значительно превосходит качество распознавания сущностей, связанных с заболеваниями – ADR, DI и Finding. Данное наблюдение может объясняться меньшей длиной сущностей, связанных с лекарствами. Так, средняя длина сущностей Drugclass, Drugform и Drugname в корпусе RuDReC составляет 1,06 слов, а сущностей ADR, DI и Finding – 1,77. Таким образом, распознавание сущностей последних трех типов представляет собой более трудную задачу, поскольку такие сущности зачастую являются словосочетаниями. Наконец, результаты экспериментов показали, что использование англоязычных данных в процессе обучения модели приводит не к повышению, а понижению качества распознавания с точки зрения макро F-меры. Тем не менее, модель RuDR-BERT, последовательно обученная на PsyTAR и RuDReC, показала прирост F-меры при распознавании сущностей побочных эффектов (+2,1%).

Табл. 4. Оценки F-меры предобученных моделей BERT на задаче извлечения именованных сущностей корпуса RuDReC

Table 4. Performance of pretrained BERT models on the RuDReC corpus named entity recognition task

Модель	**ADR**	**DI**	**Finding**	**Drugclass**	**Drugform**	**Drugname**	**Макро F-мера**
RuBERT	54.51	69.43	27.87	92.78	95.72	92.11	72.07
Multi-BERT	54.65	67.63	25.75	92.36	94.89	91.05	71.06
RuDR-BERT	60.36	72.33	**33.31**	**94.12**	95.89	93.08	**74.85**

EnRuDR-BERT	61.11	**72.84**	27.67	94.08	**96.20**	92.18	74.01
RuDR-BERT, CADEC+RuDReC	57.74	71.43	28.94	93.08	95.38	93.01	73.26
RuDR-BERT, PsyTAR+RuDReC	**62.48**	71.78	28.60	92.67	95.28	**93.17**	74.00

6. Заключение

В результате данной работы были получены следующие основные результаты. Во-первых, было исследовано влияние предметной области текстовой коллекции, использованной для предобучения модели BERT, на качество решения двух задач на русском языке: многоклассовой классификации предложений и распознавания именованных сущностей названий лекарств, показаний к применению и побочных эффектов. В ходе экспериментов была проведена оценка эффективности двух моделей архитектуры BERT: (i) RuDR-BERT, предобученной на неразмеченной части русскоязычного корпуса отзывов о лекарственных препаратах RuDReC и (ii) EnRuDR-BERT, преобученной на объединении RuDReC с англоязычным неразмеченным корпусом пользовательских отзывов медицинской тематики. В ходе экспериментов было показано, что данные модели превосходят как многоязыковую модель Multi-BERT, так и русскоязычную модель RuBERT, обученную на текстах общей предметной области. Таким образом, результаты проведенных экспериментов позволяют сделать вывод об эффективности переноса знаний путем предобучения языковых моделей BERT на неразмеченных данных медицинской предметной области.

Во-вторых, была исследована возможность использования англоязычных данных для улучшения качества решения рассмотренных задач на русском языке. Результаты показали, что последовательное обучение сначала на англоязычных данных, а затем на русскоязычных не приводит к существенному общему падению качества с точки зрения макро F-меры, при этом для отдельных типов сущностей наблюдается даже улучшение качества. Так, в задаче извлечения именованных сущностей, соответствующих побочным эффектам лекарств, последовательное обучение на англоязычном корпусе PsyTAR и корпусе RuDReC привело к увеличению F-меры на 2,1% по сравнению с обучением только на корпусе RuDReC.

В-третьих, было проведено сравнение моделей RuDR-BERT и EnRuDR-BERT при обучении только на англоязычных данных с последующей оценкой качества на корпусе RuDReC. В рамках данного эксперимента двуязыковая модель EnRuDR-BERT превзошла русскоязычную модель RuDR-BERT на 7,3% макро F-меры в задаче классификации предложений корпуса RuDReC, в то время как при обучении на RuDReC превосходство EnRuDR-BERT составляет лишь 1% макро F-меры. Данное наблюдение позволяет сделать вывод об эффективности использования размеченных данных одного языка при решении схожей задачи для другого языка, вовсе не имеющего размеченного корпуса.

Список литературы / References

[1]. Huang C.C., Lu Z. Community challenges in biomedical text mining over 10 years: success, failure and the future. Briefings in bioinformatics, vol. 17, no. 1, 2016, pp. 132-144.

[2]. Vaswani A., Shazeer N. et al. Attention is all you need. In Proc. of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 6000-6010.

[3]. Devlin J., Chang M. 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, volume 1 (Long and Short Papers), 2019, pp. 4171-4186.

[4]. Conneau A., Lample G. Cross-lingual language model pretraining. Advances in Neural Information Processing Systems, vol. 32, 2019, pp. 7059-7069.

[5]. Lample G., Conneau A. et al. Unsupervised Machine Translation Using Monolingual Corpora Only. In Proc. of the International Conference on Learning Representations, 2018, 14 p.

[6]. Artetxe M., Schwenk H. Margin-based Parallel Corpus Mining with Multilingual Sentence Embeddings. In Proc. of the 57th Annual Meeting of the Association for Computational Linguistics, 2019, pp. 3197-3203.

[7]. Tutubalina E., Alimova I. et al. The Russian Drug Reaction Corpus and neural models for drug reactions and effectiveness detection in user reviews. Bioinformatics, vol. 37, issue 2, 2021, pp. 243-249.

[8]. Alvaro N., Miyao Y., Collier N. TwiMed: Twitter and PubMed comparable corpus of drugs, diseases, symptoms, and their relations. JMIR public health and surveillance, vol. 3, issue 2, 2017, article id. e6396.

[9]. Zolnoori M. et al. A systematic approach for developing a corpus of patient reported adverse drug events: a case study for SSRI and SNRI medications. Journal of biomedical informatics, vol. 90, 2019, article no. 103091.

[10]. Karimi S., Metke-Jimenez A. et al. Cadec: A corpus of adverse drug event annotations. Journal of biomedical informatics, vol. 55, 2015, pp. 73-81.

[11]. Sarker A., Belousov M. et al. Data and systems for medication-related text classification and concept normalization from Twitter: insights from the Social Media Mining for Health (SMM4H)-2017 shared task. Journal of the American Medical Informatics Association, vol. 25, issue 10, 2018, pp. 1274-1283.

[12]. Moreno I., Boldrini E. et al. Drugsemantics: a corpus for named entity recognition in spanish summaries of product characteristics. Journal of biomedical informatics, vol. 72, 2017, pp. 8-22.

[13]. Névéol A., Anderson R.N. et al. CLEF eHealth 2017 Multilingual Information Extraction Task Overview: ICD10 Coding of Death Certificates in English and French. CLEF 2017 Working Notes. CEUR Workshop Proceedings, vol. 1866, 2017, 17 p.

[14]. Névéol A. et al. CLEF eHealth 2018 Multilingual Information Extraction Task Overview: ICD10 Coding of Death Certificates in French, Hungarian and Italian. CLEF 2018 Working Notes. CEUR Workshop Proceedings, vol. 2125, 2018, 18 p.

[15]. Shelmanov A.O., Smirnov I.V., Vishneva E.A. Information extraction from clinical texts in Russian. In Computational Linguistics and Intellectual Technologies: Papers from the Annual International Conference "Dialogue", issue 14, 2015, pp. 560-572.

[16]. Miftahutdinov Z., Sakhovskiy A., Tutubalina E. Kfu nlp team at smm4h 2020 tasks: Cross-lingual transfer learning with pretrained language models for drug reactions. In Proc. of the Fifth Social Media Mining for Health Applications Workshop & Shared Task, 2020, pp. 51-56.

[17]. Gusev A., Kuznetsova A. et al. Bert implementation for detecting adverse drug effects mentions in russian In Proc. of the Fifth Social Media Mining for Health Applications Workshop & Shared Task, 2020, pp. 46-50.

[18]. Alimova I., Tutubalina E. et al. A Machine learning approach to classification of drug reviews in Russian. In Proc. of the Ivannikov ISPRAS Open Conference, 2017, pp. 64-69.

[19]. Klein A., Alimova I. et al. Overview of the fifth social media mining for health applications (# smm4h) shared tasks at coling 2020. In Proc. of the Fifth Social Media Mining for Health Applications Workshop & Shared Task, 2020, pp. 27-36.

[20]. Magge A., Klein A. et al. Overview of the sixth social media mining for health applications (# smm4h) shared tasks at NAACL 2021. In Proc. of the Sixth Social Media Mining for Health (# SMM4H) Workshop and Shared Task, 2021, pp. 21-32.

[21]. Kuratov Y., Arkhipov M. Adaptation of deep bidirectional multilingual transformers for Russian language. arXiv preprint arXiv:1905.07213, 2019.

[22]. Тутубалина Е. В., Мифтахутдинов З. Ш. и др. Идентификация лекарственных средств со схожим терапевтическим действием на основе семантического анализа текстов. Известия академии наук. Серия химическая, no. 11, 2017 г., стр. 2180-2189 / Tutubalina E.V., Miftahutdinov Z. Sh. et al. Using semantic analysis of texts for the identification of drugs with similar therapeutic effects. Russian Chemical Bulletin, vol. 66. issue 11, 2017, pp. 2180-2189.

Информация об авторах / Information about authors

Елена Викторовна ТУТУБАЛИНА – кандидат физико-математических наук, научный сотрудник НУЛ "Моделей и методов вычислительной прагматики" факультета компьютерных наук Высшей школы экономики; старший научный сотрудник НИЛ "Хемоинформатика и молекулярное моделирование" Казанского федерального университета; исполнительный директор по исследованию данных в Sber AI. Сфера научных интересов: обработка текстов на естественном языке, извлечение информации из текстов, распознавание именованных сущностей, классификация текстов.

Elena Viktorovna TUTUBALINA – Candidate of Physical and Mathematical Sciences, Researcher at the "Models and Methods of Computational Pragmatics" Research Laboratory of the Faculty of Computer Science, Higher School Economics; Senior Researcher at the "Chemoinformatics and Molecular Modeling" Research Laboratory, Kazan Federal University; Executive Director of Data Science at Sber AI. Her research interests include natural language processing, information extraction, named entity recognition, text classification.

Андрей Сергеевич САХОВСКИЙ – лаборант НИЛ "Хемоинформатика и молекулярное моделирование" Казанского федерального университета; студент 1 курса кафедры математических методов прогнозирования факультета вычислительной математики и кибернетики Московского государственного университета. Сфера научных интересов: обработка текстов на естественном языке, классификация текстов, анализ текстов социальных сетей.

Andrey Sergeyevich SAKHOVSKIY – Laboratory Assistant at the "Chemoinformatics and Molecular Modeling" Research Laboratory of the Kazan Federal University; 1st-year graduate student of the Department of Mathematical Forecasting Methods, Faculty of Computational Mathematics and Cybernetics, Moscow State University. His research interests include natural language processing, text classification, social media text analysis.

DOI: 10.15514/ISPRAS-2021-33(6)-16

Упрощенные кинетические модели горения метана для расширения возможностей пакета OpenFOAM и физико-химических библиотек

[1] *Д.С Кононов, ORCID: 0000-0001-9695-2820 <dr.kononoff@yandex.ru>*
[1] *В.Ю. Гидаспов, ORCID: 0000-0002-5119-4488 <gidaspov@mai.ru>*
[2] *С.В. Стрижак, ORCID: 0000-0001-5525-5180 <s.strijhak@ispras.ru>*
[1] *Московский авиационный институт,*
125993, г. Москва, Волоколамское шоссе, д. 4
[2] *Институт системного программирования им. В.П. Иванникова РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Рассмотрены сокращенные механизмы горения углеводородных топлив, на их основе проведено расширение возможностей пакета OpenFOAM и физико-химических библиотек, применимых для численного моделирования процессов, протекающих в метано-воздушных смесях. На примере метана рассматривается модифицированный механизм горения углеводородного топлива. Выбор данного вещества обусловлен перспективностью и практическим интересом к данному виду топлива в настоящее время. Осуществляется сравнение результатов, полученных в результате применения решателей, созданных в МАИ и ИСП РАН. Приведены физико-математическая модель, численные алгоритмы и результаты расчетов нестационарных физико-химических процессов, протекающих в метано-воздушных смесях. Проводится сравнение процессов эволюции значений температуры и концентраций химических компонент при постоянном давлении и энтальпии, оцениваются время задержки воспламенения и уровень значений величин при достижении состояния термодинамического равновесия. Рассмотрен процесс течения метано-воздушной смеси в трубе с отражением набегающей на стенку ударной волны. Численно решаются нестационарные уравнения газовой динамики, дополненные уравнениями химической кинетики. Эффекты вязкости, теплопроводности и диффузии не учитываются. Получены и проанализированы распределения параметров течения за отраженной ударной волной. Проиллюстрировано распространение детонационной волны в колебательном режиме. Показана согласованность результатов расчетов используемых решателей. Даны оценки дальнейшего возможного применения данного сокращенного механизма горения.

Ключевые слова: горение метана; механизмы химических реакций; OpenFOAM; физико-химические библиотеки; reactingPimpleCentralFoam; численное моделирование; смесь совершенных газов

Для цитирования: Кононов Д.С., Гидаспов В.Ю., Стрижак С.В. Упрощенные кинетические модели горения метана для расширения возможностей пакета OpenFOAM и физико-химических библиотек. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 229-240. DOI: 10.15514/ISPRAS–2021–33(6)–16

Simplified kinetic models of methane combustion to expand the capabilities of the OpenFOAM package and physicochemical libraries

[1] *D.S. Kononov, ORCID: 0000-0001-9695-2820 <dr.kononoff@yandex.ru>*
[1] *V.Yu. Gidaspov, ORCID: 0000-0002-5119-4488 <gidaspov@mai.ru>*
[2] *S.V. Strijhak, ORCID: 0000-0001-5525-5180 <s.strijhak@ispras.ru>*
[1] *Moscow Aviation Institute,*
4,Volokolamskoe shosse, Moscow, 125993, Russia
[2] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. Simplified mechanisms of combustion of hydrocarbon fuels are considered, on their basis the expansion of the capabilities of the OpenFOAM package and physicochemical libraries, applicable for the numerical simulation of processes occurring in methane-air mixtures, is carried out. A modified mechanism of combustion of hydrocarbon fuel is investigated. The choice of this substance is due to the prospects and practical interest in this type of fuel at the present time. Compares the results obtained by using the solvers created at the MAI and ISP RAS. A physical and mathematical model, numerical algorithms and results of calculations of non-stationary physical and chemical processes occurring in methane-air mixtures are presented. A comparison is made of the values of temperature and concentration of the chemical at constant pressure and enthalpy, the ignition time and the level of values of in are estimated when the state of thermodynamic equilibrium is reached. The process of flow of a methane-air mixture in a tube with reflection of a shock wave incident on the wall is considered. The unsteady equations of gas dynamics are solved numerically, supplemented by the equations of chemical kinetics. The effects of viscosity, thermal conductivity and diffusion are not taken into account. The distributions of the flow parameters behind the reflected shock wave are obtained and analyzed. The propagation of a detonation wave in an oscillatory mode is illustrated. The consistency of the calculation results of the solvers used is shown. Estimates of the possible application of this reduced combustion mechanism are given.

Keywords: methane combustion; mechanisms of chemical reactions; OpenFOAM; physicochemical libraries; reactingPimpleCentralFoam; numerical simulation; mixture of perfect gases

For citation: Kononov D.S., Gidaspov V.Y. Strijhak S.V. Simplified kinetic models of methane combustion to expand the capabilities of the OpenFOAM package and physicochemical libraries. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 229-240 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–16

1. Введение

В настоящее время большой практический и научный интерес представляет изучение горения метано-воздушных смесей. Данный вид топлива рассматривается как в контексте использования в перспективных энергетических установках различного назначения, так и в задачах, связанных с обеспечением безопасности труда человека и его жизнедеятельности. Для моделирования процессов, протекающих в метано-воздушных смесях, применяются как детальные кинетические механизмы, так и глобальные брутто-механизмы [1-3].

В данной работе предлагается расширение функционала, предоставляемого открытой библиотекой OpenFOAM и двумя встроенными библиотеками для моделирования физико-химических процессов, для возможности использования модифицированного кинетического механизма окисления метана [4].

В качестве первой тестовой задачи берется стандартный решатель chemFoam, предназначенный для демонстрации протекания химических реакций в расчетной области, состоящей из одной ячейки. В качестве критериев сравнения берутся задержка воспламенения и распределение параметров при достижении состояния термодинамического равновесия. Второй тестовой задачей является исследование течения в ударной трубе после

отражения ударной волны от запаянного торца. Ее численный расчет проводится с использованием разработанного в ИСП РАН решателя reactingPimpleCentralFoam и сравнивается с результатами, полученными путем применения сеточно-характеристического метода и метода Годунова в одномерной постановке в решателях МАИ.

2. Математическая модель

2.1 Модель химической кинетики

Рассматривается многокомпонентная система переменного состава из N веществ, в которых протекает N_R реакций вида:

$$\sum_{i=1}^{N} \overleftarrow{\nu}_i^{(r)} M_i \underset{\overrightarrow{W}^{(r)}}{\overset{\overleftarrow{W}^{(r)}}{\rightleftarrows}} \sum_{i=1}^{N} \vec{\nu}_i^{(r)} M_i, \quad \overleftarrow{q}^{(r)} = \sum_{i=1}^{N} \overleftarrow{\nu}_i^{(r)}, r = 1..N_r, \tag{1}$$

где r – порядковый номер реакции , $\overleftarrow{\nu}_i^{(r)}$,– стехиометрические коэффициенты, $\overleftarrow{q}^{(r)}$ – порядок соответсвующих элементарных реакций, $\overleftarrow{W}^{(r)}$ – скорость (r)-ой химической реакции в прямом и обратном направлениях.

Скорость химической реакции прямо пропорциональна произведению объемных концентраций участвующих в ней компонентов и константы скорости реакции $\overleftarrow{K}^{(r)}$:

$$\overleftarrow{W}^{(r)} = \overleftarrow{K}^{(r)} \prod_i \prod (\rho\gamma_i)^{\overrightarrow{\nu_i}^{(r)}}, \left[\frac{\text{моль}}{\text{с} \cdot \text{м}^3}\right], \tag{2}$$

где ρ – плотность, γ_i – мольно-массовая концентрация i-того компонента смеси.

Константа скоростей прямых реакций рассчитывается по обобщенной формуле Аррениуса:

$$\vec{K}^{(r)} = A\left(\frac{T}{T_0}\right)^n \left(\frac{P}{P_0}\right)^m \exp\left(-\frac{E}{T}\right), \left[\frac{(\text{моль}/\text{м}^3)^{\vec{q}^{(r)}-1}}{\text{с}}\right], \tag{3}$$

где A, n, m, E - постоянные величины, индивидуальные для каждой реакции, T – температура смеси.

2.2 Модель термодинамики

Рассматривается смесь совершенных газов, термодинамические свойства которой описывались путем задания выражения для потенциала Гиббса [5]:

$$G(P, T, \bar{\gamma}) = \sum_{i=1}^{N} \gamma_i \left(G_i^0(T) + RT \ln \frac{P_i}{P_0}\right), \tag{4}$$

где R – универсальная газовая постоянная, P_0– нормальное давление, $G_i^0(T)$ – известные зависимости [5], температурная часть молярного потенциала Гиббса отдельного компонента смеси. Внутренняя энергия, плотность смеси выражаются через потенциал Гиббса и его частные производные:

$$E(T, \gamma_i) = G - T\left(\frac{\partial G}{\partial T}\right)_P - P\left(\frac{\partial G}{\partial P}\right)_T = \sum_{i=1}^{N} \gamma_i E_i(T), \tag{5}$$

$$\rho(P, T, \bar{\gamma}) = \frac{1}{\left(\frac{\partial G}{\partial P}\right)_T} = \frac{P}{RT\sum_{i=1}^{N}\gamma_i}. \tag{6}$$

3. Особенность реализации

Особенностью реализации является доработка имеющегося в стандартном пакете модуля, поскольку имеющиеся библиотеки в OpenFOAM (libspecie и libreactionThermophysicalModels) поддерживает расчет констант скоростей химических реакций как

$$\vec{K}^{(r)} = AT^n \exp\left(-\frac{E}{T}\right), \left[\frac{(\text{моль}/\text{м}^3)^{\vec{q}^{(r)}-1}}{\text{с}}\right], \quad (7)$$

что требует доработки как расчетного метода, так и способов задания/считывания параметров, характеризующих химическую реакцию.

4. Постановка задач

Итоговый кинетический механизм, полученный по результатам модификации [6], может быть представлен как табл. 1.

Табл.1 Кинетический механизм горения метана
Table 1 Methane combustion kinetic mechanism

Реакция	A, $\frac{(\text{моль}/\text{м}^3)^{\vec{q}^{(r)}-1}}{\text{с}}$	n	m	E, K
$CH_4 + O_2 \Leftrightarrow 2/3CO + 4/3H_2O + 1/3CH_4$	6.0E+8	0	-0.2264	1.88406E+5
$H_2+H_2+O_2 \Leftrightarrow H_2O+H_2O$	7.0E+7	0	-0.5	8.8E+4
$CO+CO+O_2 \Leftrightarrow CO_2+CO_2$	8.5E+6	0	-1.5	8.8E+4
$CO_2+H_2 \Leftrightarrow CO+H_2O$	1.0E+9	0	-1	1.74E+5

Полагалось, что реакции протекают в метано-воздушной смеси.

С данным кинетическим механизмом проведены расчеты в двух задачах.

4.1 Моделирование течения химической реакции при постоянных давлении и энтальпии

Предполагается, что система замкнута, однородна по пространству, $p = const, H = const$. Необходимо определить временные зависимости концентраций химических компонентов, плотности и температуры.

Искомыми параметрами являются концентрации и температура, которые могут быть найдены из системы дифференциально-алгебраических уравнений

$$\rho \frac{d\gamma_i}{dt} = W_i(\rho, T, \gamma_1, \ldots, \gamma_N), \gamma_i(0) = \gamma_i^0, i = 1, \ldots, N,$$
$$\rho(P, T, \bar{\gamma}) = \frac{P}{RT\sum_{i=1}^{N}\gamma_i}, \quad H = \sum_{i=1}^{N}\gamma_i H_i(T).$$

Данная задача является отправной точкой для верификации привнесенных изменений в физико-химические библиотеки. В качестве эталонных данных берутся распределения, полученные в ходе проведения расчета в программном комплексе, представленном в работе [6].

4.2 Моделирование течения за отраженной ударной волной

Рассматривается течение в ударной трубе, заполненной горючей метано-воздушной смесью, возникающее после отражения ударной волны (УВ) от закрытого торца ударной трубы. Для описания течения в областях непрерывности используются уравнения физической газовой динамики:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \left(\rho u_j\right)}{\partial x_j} = 0. \tag{8}$$

$$\frac{\partial (\rho u_i)}{\partial t} + \frac{\partial \left(\rho u_i u_j\right)}{\partial x_j} = -\frac{\partial P}{\partial x_j} + \frac{\partial \tau_{ij}}{\partial x_j}, \tag{9}$$

$$\frac{\partial (\rho h_s)}{\partial t} + \frac{\partial \left(\rho h_s u_j\right)}{\partial x_j} + \frac{\partial (\rho K)}{\partial t} + \frac{\partial \left(\rho K u_j\right)}{\partial x_j} - \frac{\partial P}{\partial t} =$$

$$= \frac{\partial \left(u_i \tau_{ij}\right)}{\partial x_j} + \frac{\partial}{\partial x_j}\left(\chi \frac{\partial h_s}{\partial x_j}\right) + \sum_{m=1}^{N} \frac{\partial}{\partial x_j}\left(\chi h_{s,m} \frac{\partial Y_m}{\partial x_j}\right) + S_{hc}, \tag{10}$$

$$\frac{\partial (\rho k)}{\partial t} + \frac{\partial \left(\rho k u_j\right)}{\partial x_j} = \frac{\partial}{\partial x_j}\left((\mu + \sigma_K \mu_t)\frac{\partial k}{\partial x_j}\right) + \tilde{P}_K - \rho \beta^* k \omega, \tag{11}$$

$$\frac{\partial (\rho \omega)}{\partial t} + \frac{\partial \left(\rho \omega u_j\right)}{\partial x_j} = \frac{\partial}{\partial x_j}\left((\mu + \sigma_\omega \mu_t)\frac{\partial \omega}{\partial x_j}\right) + \alpha \rho S^2 - \rho \beta \omega^2 + (1 - F_1) 2 \rho \sigma_{\omega 2} \frac{1}{\omega}\frac{\partial \omega}{\partial x_j}\frac{\partial k}{\partial x_j}, \tag{12}$$

$$\frac{\partial (\rho Y_m)}{\partial t} + \frac{\partial \left(\rho Y_m u_j\right)}{\partial u_j} = \frac{\partial}{\partial x_j}\left((\mu + \mu_t)\frac{\partial Y_m}{\partial x_j}\right) + W_m \quad m = 1 \ldots N - 1 \quad \sum_{m=1}^{N} Y_m = 1, \tag{13}$$

$$P = \left(\sum_{m=1}^{N} \frac{Y_m}{M_m}\right) \rho R T, \tag{14}$$

$$h_s = \sum_{m=1}^{N} Y_m \, h_{s,m}, \tag{15}$$

$$h_{s,m} = \int_{T_{std}}^{T} C_{p,m} dT \quad m = 1 \ldots N, \tag{16}$$

$$h_c = \sum_{m=1}^{N} Y_m h_{c,m}, \tag{17}$$

$$\tau_{ij} = (\mu + \mu_t)\left[\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}\delta_{ij}\frac{\partial u_k}{\partial x_k}\right], \tag{18}$$

$$S_{hc} = -\sum_{m=1}^{N} W_m h_{c,m}, \tag{19}$$

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right), S^2 = S_{ij} S_{ij}. \tag{20}$$

Здесь ρ – плотность, u_j – скорость, P – давление, T– температура, h_s – энтальпия, $h_{c,m}$ – энтальпия образования m-ого компонента при $T=T_0$, $h=E+P/\rho$, τ_{ij} – тензор напряжений, Y_m, M_m – массовая концентрация и молекулярный вес mой компоненты вещества, K – удельная кинетическая энергия, W_m – скорости химических реакций, μ , μ_t – молекулярная и турбулентная вязкости, k и ω – турбулентная кинетическая энергия и удельная скорость диссипации турбулентной энергии для $k-\omega$ SST модели турбулентности, S_{ij} - тензор деформаций-напряжений, χ – коэффициент диффузии, N – количество компонент смеси.

При отсутствии явлений вязкости, теплопроводности и диффузии величины τ_{ij}, χ, μ , μ_t, k, ω, положены равными нулю.

На ударной волне выполняются соотношения Ренкина-Гюгонио:

$$\begin{cases} \rho_1 v_1 = \rho_2 v_2, \\ P_1 + \rho_1 v_1^2 = P_2 + \rho_2 v_2^2, \\ E_1 + \dfrac{P_1}{\rho_1} + \dfrac{v_1^2}{2} = H_2 + \dfrac{P_2}{\rho_2} + \dfrac{v_2^2}{2}, \\ \bar{\gamma}_1 = \bar{\gamma}_2 \end{cases} \quad (21)$$

где индексом «2» помечены величины после ударной волны, индексом «1» - до, $v = D - u$, D – скорость распространения разрыва.

5. Результаты расчета

Решение задачи моделирования протекания химических реакций при постоянных значениях давления и энтальпии возможно с помощью применения стандартного решателя chemFoam в составе библиотеки OpenFOAM v1912 [7].

Поскольку расчетная область в нем состоит только из одной ячейки, единственным фактором, влияющим на эволюцию концентраций веществ и значения температуры, являются химические реакции.

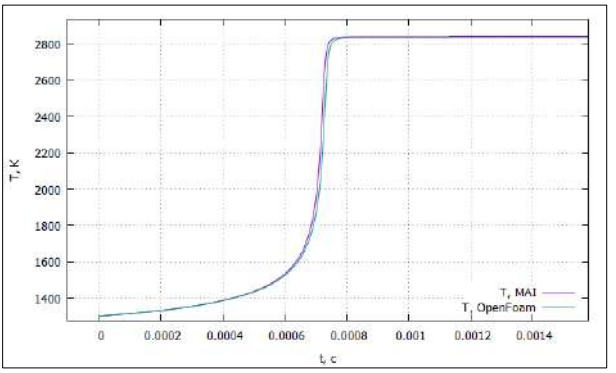


Рис. 1. Сравнение распределения температур
Fig. 1. Temperature comparison

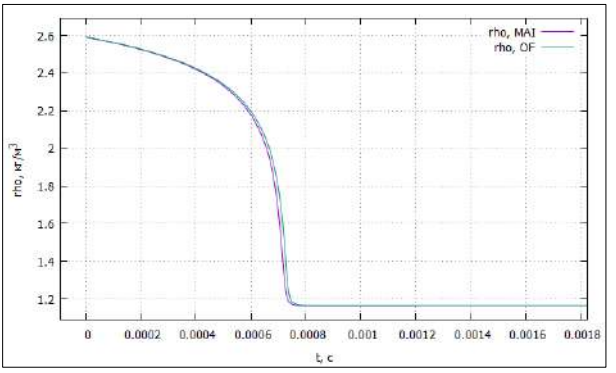


Рис. 2. Сравнение распределения плотностей
Fig. 2. Density comparison

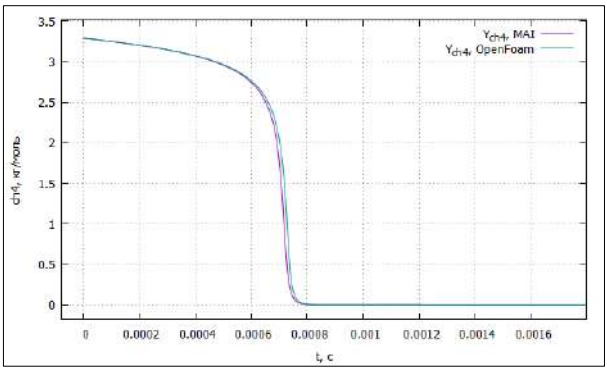


Рис. 3. Сравнение мольно-массовых концентраций метана
Fig. 3. Mol-mass fraction of methane comparison

При сравнении задержек воспламенения и температуры при достижении состояния термодинамического равновесия наблюдается хорошее согласование (рис. 1-3). В качестве начальных данных брались T = 1300 K, p = 1012857 Па, смесь $0.091CH_4 + 0.182O_2 + 0.727N_2$. В chemFoam решателе разностная схема по времени бралась как Euler, шаг по времени принимался равным 1e-7 секунд.

Для моделирования течения в ударной трубе задавались два распределения макропараметров: до- (u = 0, T = 300 K, p = 10000 Па) и за- (u = -1006.993 м/с, T = 950.414 K, p = 152000.602 Па) координатой постановки ударной волны. Концентрации компонентов смеси ($0.091CH_4 + 0.182O_2 + 0.727N2$) на всем протяжении канала изначально полагались одинаковыми. Граничные условия задавались как неподвижная жесткая стенка:

$$\begin{cases} u_w = -u_1, \\ p_w = p_1, \\ \rho_w = \rho_1. \end{cases}$$

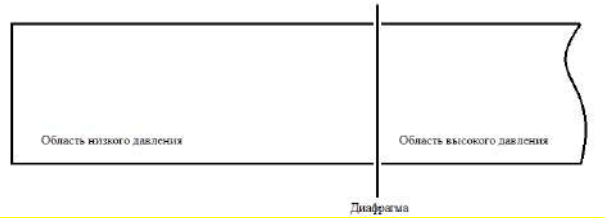
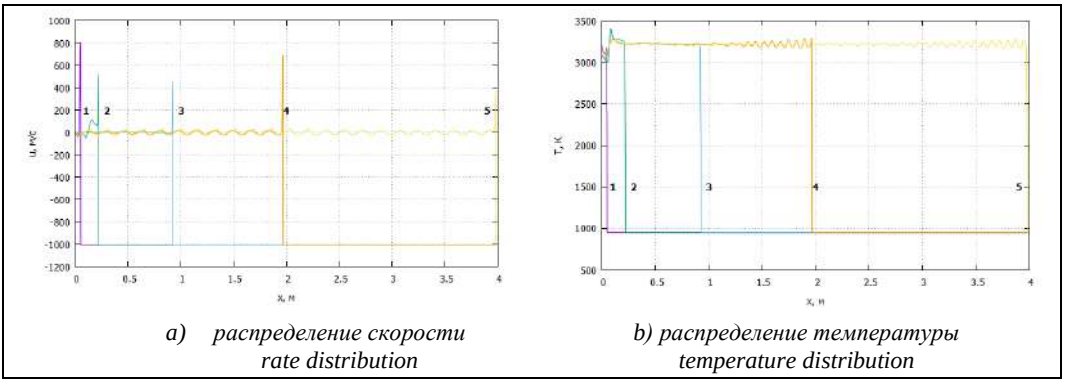


Рис. 4. Эксперимент в ударной трубе
Fig. 4. Experiment in a shock tube

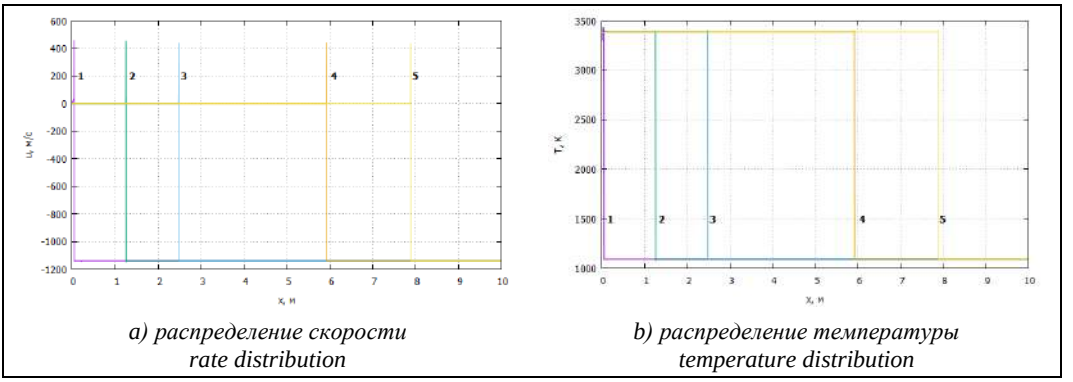
В одномерной постановке расчеты проводились с использованием сеточно-характеристического метода [8] (решатель «MAI 1») и метода Годунова [9] (решатель «MAI 2»). После удаления диафрагмы образовавшаяся ударная волна перемещалась в сторону закрытого левого торца ударной трубы (рис. 4). После отражения ударной волны от стенки наблюдался рост температуры и давления, которые приводили к воспламенению горючей смеси и образованию волны горения [6]. На некотором расстоянии от стенки происходило взаимодействие волны горения с отраженной ударной волной, в результате которого образовывалась пересжатая детонационная волна. Расчетным путем было получено, что детонационная волна распространялась в колебательном режиме при Махе падающей волны M < 3.9 (рис. 5), и в режиме с постоянной скоростью при M > 3.9 (рис. 6).



a) распределение скорости
rate distribution

b) распределение температуры
temperature distribution

Рис. 5. Колебательный режим распространения отраженной ударной волны, решатель MAI 2
Fig. 5. Parameters fluctuations of reflected shock wave, MAI 2 solver



a) распределение скорости
rate distribution

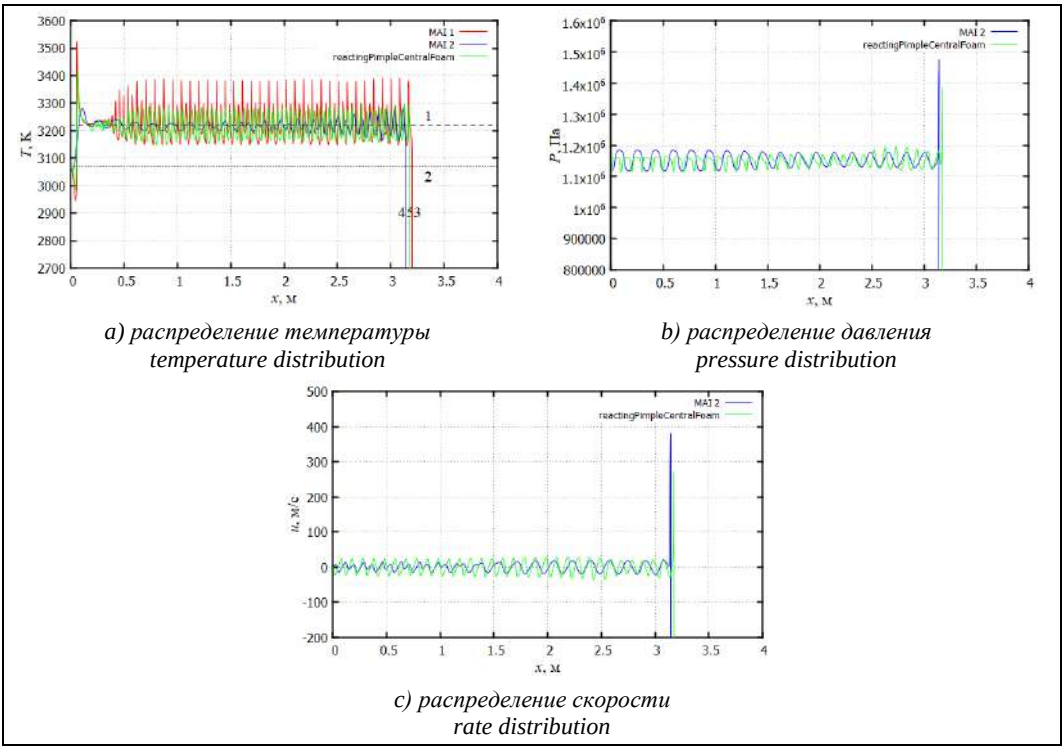
b) распределение температуры
temperature distribution

Рис. 6. Режим распространения отраженной ударной волны без колебаний, решатель MAI 2
Fig. 6. Parameters without fluctuations of reflected shock wave, MAI 2 solver

При задании параметров в решателе reactingPimpleCentralFoam, разработанного в ИСП РАН [11], также наблюдается медленное течение химических реакций до столкновения ударной волны с торцом (рис. 7). Разностная схема по времени в reactingPimpleCentralFoam бралась Euler (неявная временная схема), шаг по времени принимался равным 1e-6 секунд. Для совместного решения уравнений скорости, давления и температуры использовался алгоритм PIMPLE, для аппроксимации конвективных слагаемых использовались численные схемы vanLeer, vanLeerV (схемы ван Лира), linear (линейная интерполяция). Граничные условия для всех величин полагались zeroGradient для условия типа «стенка». После дискретизации слагаемых уравнений (8-13) получившаяся система линейных алгебраических уравнений решалась с помощью стабилизированного метода сопряженных градиентов (PBiCGStab) с предобуславливателем с моделью диагонального LU-разложения (DILU).

Размер расчетной области составляет 4 м х 2 м х 2 м. Для одномерного расчета область была разделена на 10000 х 1 х 1 ячеек, для двумерного расчета – на 10000 х 10 х 1 ячеек.

При сравнении распределения параметров в одномерных нестационарных расчетах наблюдалось хорошее согласование распределения параметров (рис. 7). Во всех трех расчетах наблюдается характерная «ячеистая» структура, колебания значений наблюдаемых параметров осуществляются вокруг значений, соответствующих равновесной отраженной детонационной волне.



a) распределение температуры
temperature distribution

b) распределение давления
pressure distribution

c) распределение скорости
rate distribution

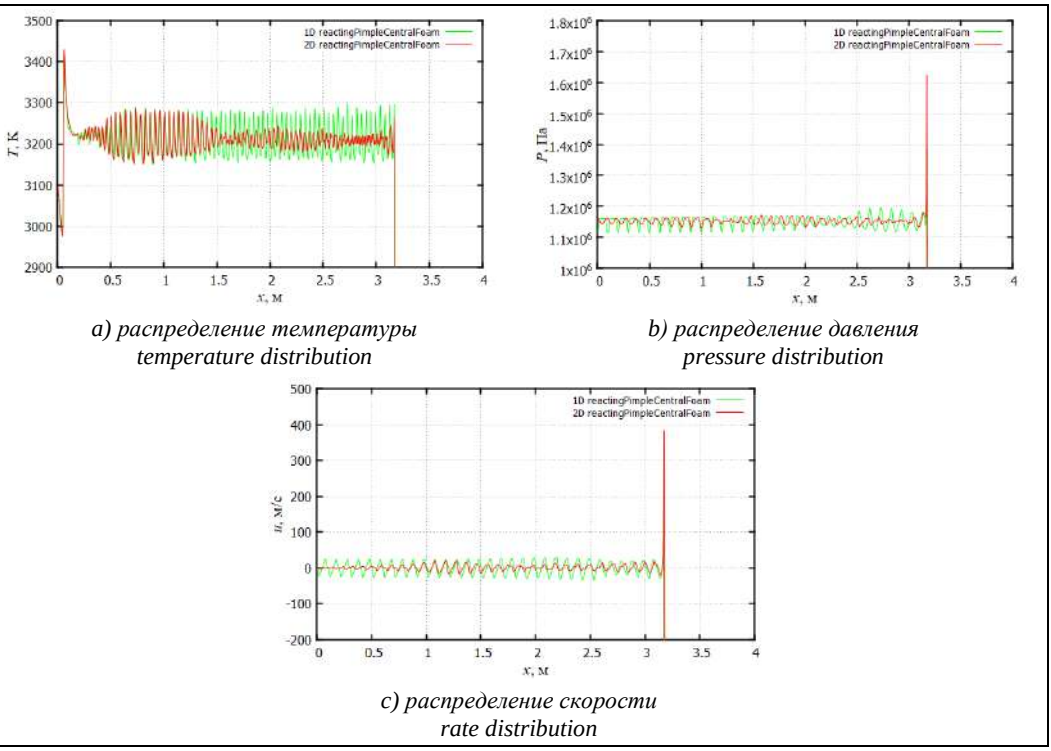
Рис. 7. 1D распределение параметров при $M_П$=3.6 при t = 0,004 с, 2 – отраженная детонационная волна, 3 – детонационная волна Чепмена-Жуге
Fig. 7. 1D parameters distribution, M_{Sh}=3.6, t = 0,004 sec, 2 – reflected detonation wave, Chapman-Jouguet detonation wave

При задании параметров в трехмерной постановке с использованием решателя reactingPimpleCentralFoam, разработанного в ИСП РАН [11], также наблюдается медленное течение химических реакций до столкновения ударной волны с торцом (рис. 7). Разностная схема по времени в reactingPimpleCentralFoam бралась Euler, шаг по времени принимался равным 1e-6 секунд. Для совместного решения уравнений скорости, давления и температуры использовался алгоритм PIMPLE, для аппроксимации конвективных слагаемых использовались численные схемы vanLeer, vanLeerV, linear. Граничные условия для всех величин полагались zeroGradient для условия типа «стенка». После дискретизации слагаемых уравнений (8-13) получившаяся система линейно-алгебраических уравнений решалась с помощью стабилизированного метода сопряженных градиентов (PBiCGStab) с предобуславливателем с моделью диагонального LU-разложения (DILU).

Размер расчетной области составляет 4 м × 2 м × 2 м. Для одномерного расчета область была разделена на 10000 × 1 × 1 ячеек, для двумерного расчета – на 10000 × 10 × 1 ячеек.

При сравнении распределения параметров в одномерных нестационарных расчетах наблюдалось хорошее согласование распределения параметров (рис. 7). Во всех трех расчетах наблюдается характерная «ячеистая» структура, колебания значений наблюдаемых параметров осуществляются вокруг значений, соответствующих равновесной отраженной детонационной волне.

При сравнении расчетов решателем reactingPimpleCentralFoam в одномерной и двумерной расчетных областях наблюдается сохранение ячеистой структуры [12] распределения и одинаковый уровень колебаний значений параметров (рис. 8).



a) распределение температуры
temperature distribution

b) распределение давления
pressure distribution

c) распределение скорости
rate distribution

Рис. 8. Распределение параметров при $M_{П}$=3.6 при t = 0,004 с, в расчетах решателем reactingPimpleCentralFoam
Fig. 8. Parameters distribution, M_{Sh}=3.6, t = 0,004 sec, reactingPimpleCentralFoam calculations

6. Заключение

В работе проведена модификация стандартной химико-физической библиотеки прикладного программного пакета OpenFOAM v1912, позволившая использовать в расчетах обобщенную форму задания химических реакций, протекающих в смесях. Опробовано применение модифицированного сокращенного глобального механизма горения метана в воздухе. Расчеты выполнялись на персональном компьютере и вычислительном кластере UniHUB ИСП РАН.

Получено согласованное распределение параметров в решателях, разработанных в МАИ и ИСП РАН. В дальнейшем рассматриваемый кинетический брутто-механизм может быть применен для проведения расчетов в других видах углеводородных топлив путем модификации реакции, содержащей метан.

Список литературы / References

[1]. Peters N. Turbulent Combustion. Cambridge University Press, 2000, 324 p.
[2]. Poinsot T., Veynante D. Theoretical and Numerical Combustion. 2nd edition. Edwards, 2005, 540 p.
[3]. Warnatz J., Maas U., W. Dibble R.W. Combustion: Physical and Chemical Fundamentals, Modeling and Simulation, Experiments, Pollutant Formation. 4th Edition. Springer, 2006, 390 p.

[4]. Басевич В.Я., Фролов С.М. Глобальные кинетические механизмы, использующиеся при моделировании многостадийного самовоспламенения углеводородов в реагирующих течениях. Химическая физика, том 25, no. 6, 2006. стр. 54-62 / Basevich V.Ya., Frolov S.M. Global kinetic mechanisms used in modeling multistage autoignition of hydrocarbons in reacting flows, Chemical Physics, vol. 25, no. 6, 2006. pp. 54-62 (in Russian).

[5]. Гурвич Л.В., Вейц И.В. и др. Термодинамические свойства индивидуальных веществ. Справочное издание в 4-х томах. М., Наука, 1979-1982 гг. / Gurvich L.V., Veyts I.V. et al. Thermodynamic properties of individual substances. Reference edition in 4 volumes. M., Nauka, 1979-1982 (in Russian).

[6]. Гидаспов В.Ю., Кононов Д.С., Северина Н.С. Моделирование воспламенения и детонации метано-воздушных смесей за отраженной ударной волной. Теплофизика высоких температур, том 58, no. 6, 2020 г., стр. 909-914 / Gidaspov V.Y., Kononov D.S., Severina N.S. Simulation of the ignition and detonation of methane-air mixtures behind a reflected shock wave. High Temperature, vol. 58, no. 6, 2020, pp. 846-851.

[7]. Weller H.G., Tabor G. et al. A tensorial approach to computational continuum mechanics using object-oriented techniques. Computers in Physics, vol. 12, 1998, pp. 620–631.

[8]. Холодов А.С. Сеточно-характеристические численные методы для многомерных задач механики сплошных сред. Вопросы кибернетики, вып. 15, 1987 г., стр. 140-163 / Kholodov A.S. Grid-characteristic numerical methods for multidimensional problems of continuum mechanics. Cybernetics Issues, issue 15, 1987, pp. 140-163 (in Russian).

[9]. Годунов С.К., Забродин А.В. и др. Численные решения многомерных задач газовой динамики. М., Наука, 1976 г., 400 стр. / Godunov S.K., Zabrodin A.V. et al. Numerical solution of multidimensional problems of gas dynamics. M., Nauka, 1976, 400 p. (in Russian).

[10]. Пирумов У.Г. Математическое моделирование в проблемах охраны воздушного бассейна. М., Изд-во МАИ, 2001 г., 340 стр. / Pirumov U.G. Mathematical modeling in the problems of air protection. Moscow, MAI Publishing House, 2001, 340 p. (in Russian).

[11]. Kraposhin M. Bovtrikova A., Strijhak S. Adaptation of Kurganov-Tadmor Numerical Scheme for Applying in Combination with the PISO Method in Numerical Simulation of Flows in a Wide Range of Mach Numbers. Procedia Computer Science. vol. 66, 2015, pp. 43-52.

[12]. Васильев А.А. Ячеистые структуры многофронтовой детонационной волны и инициирование (Обзор). Физика горения и взрыва, том 51, вып. 1, 2015 г., стр. 9-30 / Vasil'ev A.A. Cellular structures of a multifront detonation wave and initiation (Review). Combustion, Explosion, and Shock Waves, vol. 51, issue 1, 2015, pp. 1-20.

Информация об авторах / Information about authors

Дмитрий Сергеевич КОНОНОВ является инженером кафедры Вычислительной математики и программирования. Сфера научных интересов: математическое моделирование, физическая газовая динамика, химическая кинетика.

Dmitry Sergeevich KONONOV is an engineer of the Department of Computational Mathematics and Programming. Research interests: mathematical modeling.

Владимир Юрьевич ГИДАСПОВ – доктор физико-математических наук, доцент, старший научный сотрудник кафедры вычислительной математики и программирования. Сфера научных интересов: математическое моделирование, физическая газовая динамика, многофазные течения, горение и детонация.

Vladimir Yurevich GIDASPOV – Doctor of Physical and Mathematical Sciences, assistant professor, Senior Researcher of the Department of Computational Mathematics and Programming of the Moscow Aviation Institute. Research interests: mathematical modeling, physical gas dynamics, multiphase flows, combustion and detonation.

Сергей Владимирович СТРИЖАК – кандидат технических наук, ведущий инженер. Сфера научных интересов: вычислительная гидродинамика, многофазные течения, турбулентность, ветроэнергетика, параллельные вычисления.

Sergei Vladimirovich STRIJHAK – candidate of technical sciences, leading engineer. Research interests: computational fluid dynamics, multiphase flows, turbulence, wind power, parallel computing.

DOI: 10.15514/ISPRAS-2021-33(6)-17

Разработка решателя flagmanFoam для моделирования обледенения летательных аппаратов в условиях натекания мелких капель

[1] *К.А. Ватутин, ORCID: 0000-0002-4090-6320 <vatutin_kirill@ispras.ru>*
[1] *М.В. Крапошин, ORCID: 0000-0001-5730-2702 <m.kraposhin@ispras.ru>*
[2] *М.А. Кудров, ORCID: 0000-0002-7937-6005 <mkudrov@mail.ru>*
[3] *А.Б. Миллер, ORCID: 0000-0001-8521-9631 <Aleksey.Miller@tsagi.ru>*
[1,4] *В.Г. Мельникова, ORCID: 0000-0002-1280-2647 <vg-melnikova@ispras.ru>*
[2] *А.О. Морозов, ORCID: 0000-0003-4620-9662 <morozov.ao@phystech.edu>*
[1,4] *С.М. Сауткина, ORCID: 0000-0002-3140-1080 <sautkina@ispras.ru>*
[1,4] *А.А. Шевелев, ORCID: 0000-0002-5043-5796 <ashevelev@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2] *Московский физико-технический институт,*
141701, Россия, Долгопрудный, Институтский пер., д. 9
[3] *Центральный аэрогидродинамический институт им. профессора Н.Е. Жуковского,*
140180, Россия, г. Жуковский, Московская область, ул. Жуковского, 1
[4] *Московский государственный технический университет имени Н.Э. Баумана*
105005, Россия, г. Москва, ул. 2-я Бауманская, дом 5, стр. 1

Аннотация. Работа посвящена созданию программного комплекса flagmanFoam, разрабатываемого на базе пакета OpenFOAM v2012. Решатель предназначен для моделирования процессов обледенения в условиях натекания мелких капель, при характерном размере до 40 мкм, что соответствует Приложению С Авиационных правил АП-25. Приведены физико-математические модели, реализованные в решателе: для описания динамики газокапельного потока используется Эйлер-Эйлер подход , термодинамическая модель Майерса используется для описания процесса нарастания жидкой пленки и льда, для движения межфазной поверхности используется Coupled Level Set – VoF метод, для учета взаимодействия между жидкостью и обтекаемым телом используется метод погруженных границ, турбулентная вязкость вычисляется с помощью *k*-ω SST модели турбулентности. Представлены результаты моделирования на тестовых задачах и сравнение с экспериментальными данными.

Ключевые слова: обледенение; переохлажденные капли; нарастание льда; моделирование; многофазная среда; поверхность.

Для цитирования: Ватутин К.А., Крапошин М.В., Кудров М.А., Миллер А.Б. Мельникова В.Г., Морозов А.О., Сауткина С.М., Шевелев А.В. Разработка решателя flagmanFoam для моделирования обледенения летательных аппаратов в условиях натекания мелких капель. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 241-252. DOI: 10.15514/ISPRAS–2021–33(6)–17

Благодарности: Работа выполнена в рамках проекта ICE GENESIS при финансовой поддержке Министерства промышленности и торговли РФ, уникальный идентификатор проекта 824310.

Development of the flagmanFoam Solver for Modeling Aircraft Icing in Conditions of Small Droplet Inflow

[1] *K.A. Vatutin, ORCID: 0000-0002-4090-6320 <vatutin_kirill@ispras.ru>*
[1] *M.V. Kraposhin, ORCID: 0000-0001-5730-2702 <m.kraposhin@ispras.ru>*
[2] *M.A. Kudrov, ORCID: 0000-0002-7937-6005 < mkudrov@mail.ru>*
[3] *A.B. Miller, ORCID: 0000-0001-8521-9631 <Aleksey.Miller@tsagi.ru>*
[1,4] *V.G. Melnikova, ORCID: 0000-0002-1280-2647 <vg-melnikova@ispras.ru>*
[2] *A.O. Morozov, ORCID: 0000-0003-4620-9662 <morozov.ao@phystech.edu>*
[1,4] *S.M. Sautkina, ORCID: 0000-0002-3140-1080 <sautkina@ispras.ru>*
[1,4] *A.A. Shevelyov, ORCID: 0000-0002-5043-5796 <ashevelev@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2] *Moscow Institute of Physics and Technology,*
9, Insitutsky per., Dolgoprudny, 141701, Russia
[3] *Central Aerohydrodynamic Institute,*
1, Zhukovsky st., Zhukovsky, Moscow Region, 140180, Russia
[4] *Bauman Moscow State Technical University*
5/1, 2-nd Baumanskaya st., Moscow, 105005, Russia

Abstract. This work is devoted to the creation of the flagmanFoam software package developed on the basis of the OpenFOAM v2012 package. A solver has been developed to simulate icing in conditions of small droplet in-flow, with a characteristic droplet size of up to 40 microns, which corresponds to Appendix C of the Aviation Regulations AP-25. Computational models implemented in the solver are presented: the Euler-Euler approach is used to describe the dynamics of a gas-droplet flow, the Myers thermodynamic model is used to describe the growth of a liquid film and ice, the Coupled Level Set - VoF method is used to move the interface, to take into account the interaction between liquid and body, the immersed boundary method is used, the turbulent viscosity is calculated using the *k*-ω SST turbulence model. The results of modeling on test problems and comparison with experimental data are presented.

Keywords: icing; supercooled droplets; ice accretion; simulation; multiphase environment; surface.

For citation: Vatutin K.A., Kraposhin M.V., Kudrov M.A., Miller A.B., Melnikova V.G., Morozov A.O., Sautkina S.M., Shevelyov A.A. Development of the flagmanFoam solver for modeling aircraft icing in conditions of small droplet inflow. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 241-252 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–17

Acknowledgements. The work is supported by ICE GENESIS project with financial support from the Ministry of Industry and Trade of the Russian Federation, unique project identifier 824310.

1. Введение

Разработка способов эффективной противообледенительной защиты, методов подтверждения соответствия изделий авиационной техники для всех возможных условий обледенения является задачей государственной важности. Перед Российской Федерацией остро стоит задача как обеспечения безопасности полетов, так и сертификации продукции отечественной авиационной промышленности во всех условиях обледенения. Экспериментальному и численному моделированию процессов обледенения посвящено большое количество работ, например, [1-5].

Однако, на текущий момент отсутствует отечественное программное обеспечение, с помощью которого можно решать задачи сертификации. В настоящее время коллективом авторов разработан прототип программного комплекса, предназначенного для моделирования процессов обледенения, который включает в себя решатель для условий натекания мелких капель (Приложение С к CS-25) со средним диаметром до 40 мкм [6].

Программный комплекс разрабатывается на универсальном языке программирования C++ и основывается на базе открытой библиотеки OpenFOAM v2012.

2. Физико-математическая модель

Математическая модель предполагает, что в расчетной области присутствуют фазы: газ (воздух), переохлажденные капли воды, лед и пленка воды, в общем случае частично покрывающая поверхность льда. Криволинейная поверхность льда с пленкой воды разделяет две среды: двухфазную жидкую среду «газ + капли» и однофазную твердую среду «лед». В каждой среде решается своя система уравнений, которые связываются между собой через граничные условия.

2.1 Расчет обтекания тела (модуль aerodynamics)

Решение задачи динамики межфазного фронта проводится в медленном меняющемся аэродинамическом поле, выполняется расчет «на установление» промежуточных состояний. Течение несжимаемого газа (несущей фазы в среде «газ + капли») описывается уравнением неразрывности и уравнением баланса импульсов.

Уравнение неразрывности несжимаемой среды:

$$\nabla \cdot (\mathbf{U}) = 0, \tag{1}$$

Где $\mathbf{U}$ – скорость потока.

Уравнение баланса импульса:

$$\nabla \cdot (\mathbf{U} \times \mathbf{U}) - \nabla \cdot \boldsymbol{\sigma} = -\nabla p, \tag{2}$$

где p – кинематическое давление окружающей среды; $\mathbf{g}$ – вектор ускорения свободного падения; $\boldsymbol{\sigma} = \nu_e(\nabla \boldsymbol{U} + (\nabla \boldsymbol{U})^T) - \frac{2}{3}\nu_e \mathbf{I} \nabla \cdot \boldsymbol{U}$ тензор вязких напряжений; $\nu_e = \nu + \nu_t$ – коэффициент эффективной вязкости смеси; ν, ν_t – коэффициенты ламинарной и турбулентной кинематической вязкости окружающей среды; $\mathbf{I}$ – единичный тензор.

Для учета взаимодействия между жидкостью и обтекаемым телом используется метод погруженных границ [7]. Вводятся две раздельные сетки для расчета течения жидкости и для расчета параметров погруженной границы (области вблизи твердого тела).

Турбулентная вязкость вычисляется с помощью $k-\omega$ SST модели турбулентности, в которой решаются уравнения для кинетической энергии турбулентных пульсаций и турбулентной частоты. На поверхности исследуемого тела используются логарифмические пристеночные функции [8].

2.2 Расчет капельного потока (модуль dropletFlow)

Для расчета капельного потока используется континуальное приближение. Приняты следующие допущения:

- набегающий поток принимается двухфазным, состоящим из несущей фазы окружающей среды и капель одного диаметра, имеющие каждый свои скорости;
- столкновениями между каплями пренебрегаем;
- дискретная фаза считается лишенной собственного давления;
- вязкие силы проявляются только в несущей фазе и при взаимодействии капель с газом;
- траектории капель рассчитываются после получения поля скоростей несущей фазы; обратного влияния капли на несущую среду не оказывают.

В модели решается уравнение для осредненной объемной концентрации капель φ_k:

$$\frac{\partial \varphi_k}{\partial t} + \nabla \cdot (\mathbf{U}_k)\varphi_k = 0, \tag{3}$$

где t – время, $\mathbf{U}_k$ – абсолютная скорость капель, и уравнение импульса капель:

$$\frac{\partial \mathbf{U}_k}{\partial t} + \mathbf{U}_k \cdot \nabla(\mathbf{U}_k) - \nabla \cdot \boldsymbol{\sigma} = \mathbf{F}, \tag{4}$$

где Re_{eff} – турбулентное число Рейнольдса.

Выражение для ускорения $\mathbf{F}$, действующего на каплю, выглядит следующим образом:

$$\mathbf{F} = \mathbf{g} + \frac{C_{d,p}(\mathrm{Re}_p)}{m_p} \cdot \frac{\rho(\mathbf{U}_k - \mathbf{U})|(\mathbf{U}_k - \mathbf{U})}{2} S_p, \tag{5}$$

Где ρ – плотность окружающей среды, S_p – площадь поверхности капли, m_p – масса капли.

Коэффициент аэродинамического сопротивления капли $C_{d,p}$, зависящий от числа Рейнольдса капли Re_p, вычисляется в соответствии с моделью Путнэма:

$$C_{d,p}(\mathrm{Re}_p) = \begin{cases} {}^{24}/_{\mathrm{Re}_p}\left(1 + \frac{1}{6}\mathrm{Re}_p{}^{2/3}\right), \mathrm{Re} \le 1000; \\ 0.424, \mathrm{Re} > 1000. \end{cases} \tag{6}$$

Методика может учитывать распределение капельной фазы по размерам, путем добавления уравнений для капель каждого размера.

Массовый поток влаги, попадающий на единицу поверхности, $\dot{m}_{water}$ вычисляется с помощью нормальной к обтекаемой поверхности скоростью переохлажденных капель и их концентрацией у поверхности:

$$\dot{m}_{water} = -(\mathbf{U}_k \times \mathbf{n}) \cdot \varphi_k \cdot \rho_p,$$

Где ρ_p – плотность капель, $\mathbf{n}$ – вектор нормали к поверхности.

2.3 Термодинамическая модель Майерса (модуль phaseInterfaceVelocity)

Уравнения, используемые в термодинамической модели, решаются на двумерной сетке, построенной вдоль поверхности твердого тела, заново на каждом шаге по времени. Модель Майерса состоит из четырех основных уравнений и включает в себя следующие допущения: 1) переход из рыхлого льда к гладкому льду в процессе замерзания происходи мгновенно; 2) слои льда и воды считаются изотермическими. Подробно модель и значение коэффициентов изложено в [9].

Уравнение теплопроводности для льда:

$$\frac{\partial T}{\partial t} = \frac{k_i}{\rho_i c_{pi}} \frac{\partial^2 T}{\partial z^2}, \tag{7}$$

где T – температура льда, k_i – коэффициент теплопроводности для льда, c_i – коэффициент теплоёмкости льда, z – пространственная координата, ρ_i - плотность льда.

Уравнение теплопроводности для воды:

$$\frac{\partial \Theta}{\partial t} = \frac{k_w}{\rho_w c_w} \frac{\partial^2 \Theta}{\partial z^2}, \tag{8}$$

где Θ – температура воды, k_w – коэффициент теплопроводности для воды, ρ_w – плотность воды, c_w – коэффициент теплоёмкости воды.

Уравнение баланса массы:

$$\rho_i \frac{\partial b}{\partial t} + \rho_w \frac{\partial h}{\partial t} = \dot{m}_{ice}, \tag{9}$$

где b – толщина льда, h – толщина воды.

Уравнение баланса энергии:

$$\rho_g L_f \frac{\partial b}{\partial t} = k_i \frac{\partial T}{\partial z} - k_w \frac{\partial h}{\partial z}, \quad (10)$$

Где ρ_g – плотность льда, L_f – удельная теплота кристаллизации.

Задаются следующие граничные условия:

1) температура подложки T_s:

$$T(0,t) = T_s;$$

2) температура на границе вода-лёд равна температуре плавления льда T_f:

$$T(b,t) = \Theta(b,t) = T_f;$$

3) поток энергии на границе воды:

$$-k_w \frac{\partial \Theta}{\partial z}\Big|_{z=b+h} = (Q_c + Q_e + Q_d) - (Q_a + Q_k),$$

где $Q_c = H_{aw} \cdot [\Theta(b+h,t) - T_a] = q_c \cdot [\Theta(b+h,t) - T_a]$ – конвективный теплообмен на поверхности воды, T_a – температура воздуха, $q_c = H_{aw}$ – коэффициент переноса тепла между водой и воздухом, $Q_e = q_e \cdot [\Theta(b+h,t) - T_a]$ – потери тепла при испарении, $q_e = e_0 \chi$ – коэффициент потери тепла при испарении, e_0 – константа давления насыщенного пара, χ – коэффициент испарения, $Q_d = \dot{m}_{ice} \cdot [\Theta(b+h,t) - T_a] = q_d \cdot [\Theta(b+h,t) - T_a]$ – охлаждение набегающим капельным потоком, $Q_a = 0$ – аэродинамический нагрев (пренебрегаем), $Q_k = 0$ – кинетическая энергия падающих капель (пренебрегаем).

Соответственно, выражение для этого потока можно переписать следующим образом:

$$-k_w \frac{\partial \Theta}{\partial z}\Big|_{z=b+h} = (\Theta(b+h,t) - T_a)(q_c + q_e + q_d).$$

Уравнения теплопроводности упрощаются предположением:

$$\frac{\partial T}{\partial t} = \frac{\partial \Theta}{\partial t} = 0.$$

Тогда T и Θ представляют собой линейные функции z, и выражения для $\frac{\partial T}{\partial z}$ и $\frac{\partial \Theta}{\partial z}$ принимают вид:

$$\frac{\partial T}{\partial z} = \frac{T_f - T_s}{b};$$

$$\frac{\partial \Theta}{\partial z} = \frac{-(q_c + q_e + q_d)\big(T_f - T_a\big)}{k_w + h(q_c + q_e + q_d)}.$$

Таким образом, уравнение баланса энергии принимает вид:

$$\rho_g L_f \frac{\partial b}{\partial t} = k_i \frac{T_f - T_s}{b} + k_w \frac{(q_c + q_e + q_d)\big(T_f - T_a\big)}{k_w + h(q_c + q_e + q_d)}. \quad (11)$$

2.4 Модель движения межфазной границы (модуль phaseInterfaceFaMeshGenerator)

Принцип работы этого и следующего модуля заключается в «расщеплении» представления и переноса межфазного интерфейса между воздухом и льдом: в то время как сам интерфейс по-прежнему описывается криволинейной поверхностью, его деформация и перенос осуществляются не явно, а на основе решения уравнения материального баланса льда, сформулированного в консервативном виде. Это позволяет, с одной стороны, сохранить выполнение баланса массы льда, а с другой стороны — снизить риск появления самопересечений и других признаков вырожденности рассматриваемой поверхности.

Поверхность льда определяется по распределению переменной VoF (Volume of Fluid) на каждом шаге по времени. Движущаяся по неподвижной (эйлеровой) сетке поверхность льда

разрезает расчетные ячейки на многогранники произвольной формы, принадлежащие разным средам. Переменная объемной доли VoF a_{ice} для льда принимает значения от 0 («газ + капли») до 1 («лед»). Ячейка, в которой $0 < a_{ice} < 1$, содержит поверхность льда. Для точного вычисления положения поверхности льда используется знаковая функция уровня, которая вычисляется относительно объёмной доли a_{ice} (Coupled Level Set – VoF метод [10]).

При запуске модели функция уровня (знакопеременное поле расстояний от данной точки пространства до межфазной поверхности) вычисляется для нескольких рядов ячеек, окружающих поверхность. При решении уравнения переноса объёмной доли льда функция уровня пересчитывается через маркер льда исходя из предположения, что межфазная поверхностью находится в области, где объёмная доля равна ½. После вычисления функции уровня одним из указанных способов, используется процедура реинициализации, позволяющая получить поле функции уровня, соответствующее её уравнению.

2.5 Модель изменения ледяного нароста (модуль iceGrowth)

В первую очередь происходит перенос данных S_i и $\mathbf{U}_{in}$ с межфазной двумерной сетки на конечно-объемную сетку. Источник S_i задается строго внутри межфазной поверхности. Скорость нарастания льда в нормальном направлении $\mathbf{U}_{in}$ задается в ячейках рядом с межфазной поверхностью по обе стороны.

Поскольку поле скорости прироста поверхности льда всегда перпендикулярно к её текущему положению, то для него предлагается составить уравнение в потенциальном приближении. Уравнение для потенциала скорости изменения межфазной поверхности Φ_i:

$$-\nabla \cdot \mathbf{K} \nabla \Phi_i = 0. \tag{12}$$

Расчет коэффициента проницаемости $\mathbf{K}$:

$$\mathbf{K} = \begin{cases} \dfrac{|\mathbf{U}_{in}|}{\langle |\mathbf{U}_{in}| \rangle}, \text{где } \boldsymbol{U_i} \neq 0; \\ 0. \text{ где } \mathbf{U}_{in} = 0; \\ 1. \text{ в остальной области.} \end{cases}$$

Далее с помощью схемы MULES [12] назначается новое поле объемной доли льда.

$$\frac{\partial a_{ice}}{\partial t} + U^*_{ice} \cdot \nabla a_{ice} = 0. \tag{13}$$

3. Архитектура решателя

Решатель включает в себя программные реализации моделей, описанных в разд. 1, и организован согласно архитектуре, представленной на рис.1.

Шаг расчета начинается с построения поверхностной сетки межфазной поверхности. После этого выполняется расчет аэродинамики обтекания тела, для аппроксимации вязких напряжений в области нароста льда используется метод погруженных границ.

Далее происходит решение уравнений, описывающих движение капельного потока. После этого поля скорости, температуры окружающей среды и масса капель, достигших поверхности, передаются в модуль расчета пленки.

Расчет пленки происходит по уравнениям модели Майерса на дополнительной расчетной сетке, построенной на межфазной границе. В результате решения уравнений получаются толщины пленки воды и льда, а также скорость нарастания льда (модуль нормальной скорости движения межфазной поверхности). По значению этих параметров изменяется распределение значения переменной объемной доли льда на основной сетке, и получает новое положение поверхности межфазной границы, которое передается построителю дополнительной сетки.

Полученная сетка используется моделью пленки на следующем шаге расчета.

Применяемые принципы позволяют перенести основную часть вычислительной нагрузки на решение задачи внешней аэродинамики при медленно меняющемся профиле. При этом нагрузки на перестроение объёмной сетки существенно снижаются (за счет использования соответствующих методов).

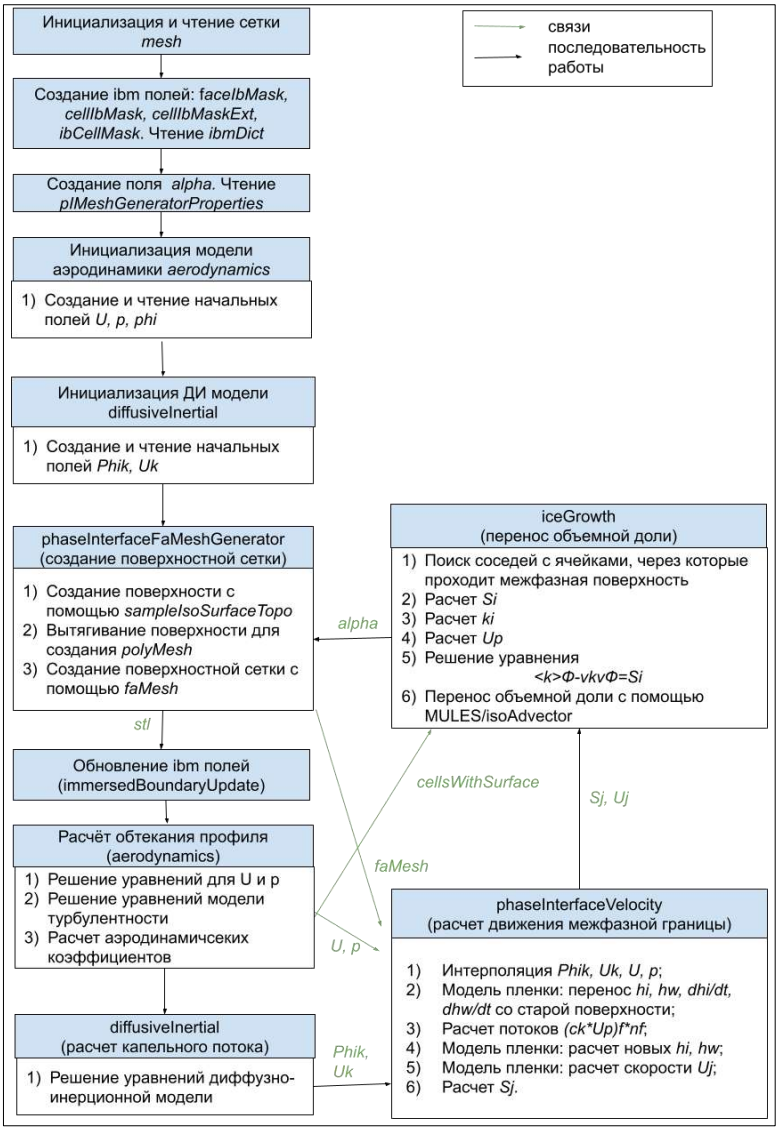


Рис. 1. Архитектура решателя flagmanFoam
Fig. 1. flagmanFoam solver architecture

Программа эффективна при параллельных вычислениях за счёт равномерности распределения вычислительной нагрузки по процессам, однотипности вычислительных операций, численных схем для всех процессов. Алгоритмы с низкой эффективностью параллельных вычислений малозатратные в последовательном режиме и изолированы в отдельные модули. В итоге эффективность параллельного масштабирования решателя становится близкой к программам, предназначенным для решения стационарных задач аэродинамики.

4. Результаты численного моделирования

4.1 Обледенение 2D цилиндра

Задача обтекания цилиндра является классической тестовой задачей для моделирования процесса обледенения. Постановка задачи обледенения цилиндра описана в [11], результаты численного моделирования сравнивались с экспериментальными данными (рис. 2). Диаметр цилиндра равнялся 15.2 см. Сетка – 79 584 ячеек, 4 уровня сгущения, шаг по времени – 12 с, время расчета на 1 ядре составило 20 ч 25 мин.

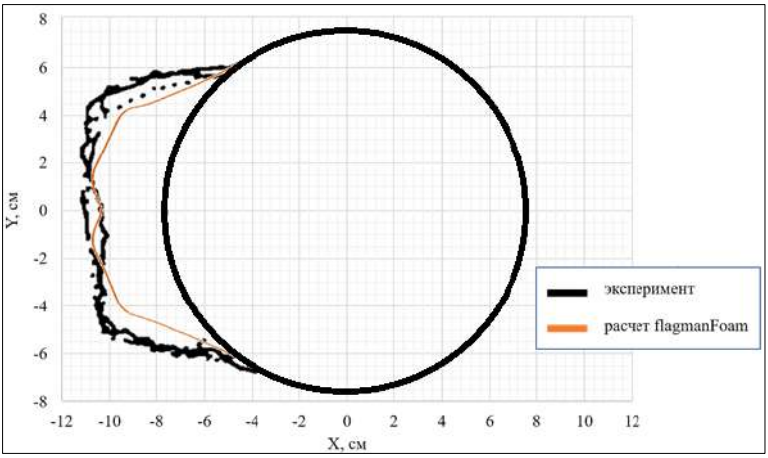


Рис. 2. Результат моделирования обледенения цилиндра (U = 76 м/с, LWC = 0.8 г/м^3, T = 260.15 K, D_p = 28 мкм, t = 18.3 мин)

Fig.2. The result of the simulation of the icing of the cylinder (U = 76 m/s, LWC = 0.8 g/m^3, T = 260.15 K, D_p = 28 μm, t = 18.3 min)

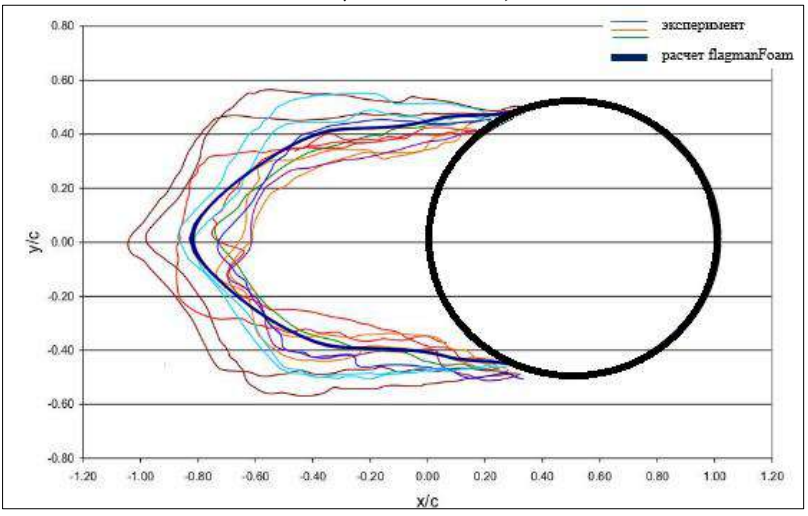


Рис. 3. Результат моделирования обледенения цилиндра (U = 67 м/с, LWC = 1 г/м3, T = 243.15 K, Dp =20 мкм, t = 10 мин)

Fig.3. The result of the simulation of the icing of the cylinder (U = 67 m/s, LWC = 1 g/m3, T = 243.15 K, Dp =20 μm, t = 10 min)

Вторая задача обтекания была решена для цилиндра диаметром 3.81 см [12]. Сетка – 74 248 ячеек. Шаг по времени – 4 с. Время расчета на 4 ядрах кластера UniHub составило 11 ч 30 мин.

4.2 Обледенение 2D профиля NACA0012

Для проверки работоспособности программного комплекса flagmanFoam была рассмотрена задача обтекания 2D профиля NACA0012 в режиме «rime ice» (рыхлый лед) [13].

Крыло размером 53.34 см находится под углом 4 градуса к потоку. Расчетная область составила 7,5 на 5 м. Сетка – 114 090 ячеек, 5 уровней разрешения. Шаг по времени – 10 с.

Результаты моделирования приведены на рис. 4 и 5. Время расчета двух случаев составило 8 ч 30 мин и 10 ч 30 мин соответственно.

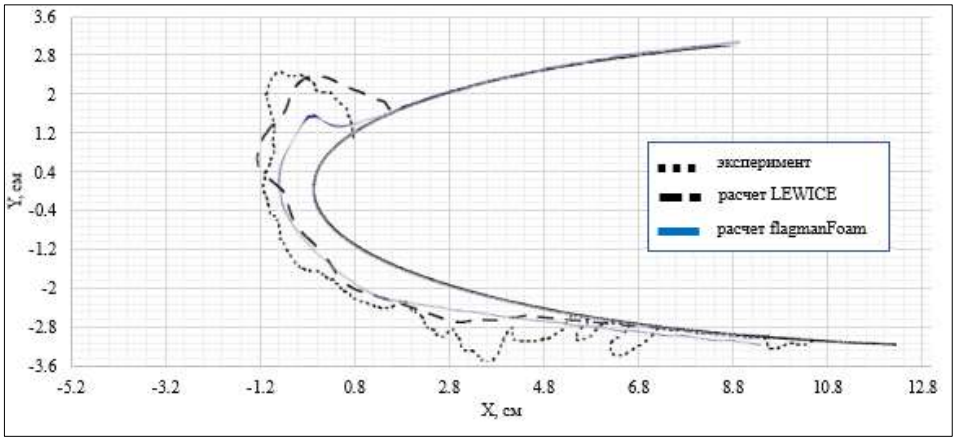


Рис. 4. Результат моделирования обледенения профиля NACA0012 (U = 67.1 м/с, LWC = 1.6 г/м^3, T = 267.59 K, D_p =30 мкм, t = 6 мин)

Fig. 4. Result of modeling icing profile NACA0012 (U = 67.1 m/s, LWC = 1.6 g/m^3, T = 267.59 K, D_p =30 μm, t = 6 min)

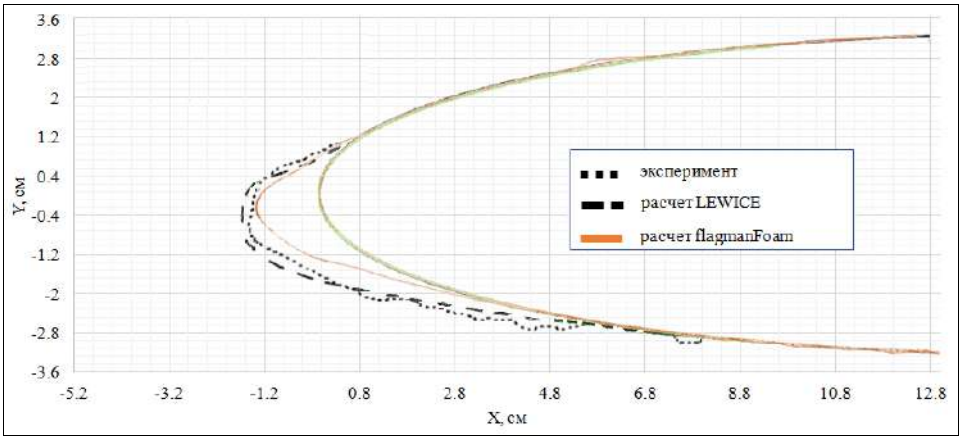


Рис. 5. Результаты моделирования профиля NACA 0012 (U = 103 м/с, LWC = 0.55 г/м3, T = 247 K, Dp = 20 мкм, t = 7 мин)

Fig. 5. Result of modeling icing profile NACA0012 (U = 103 m/s, LWC = 0.55 g/m3, T = 247 K Dp = 20 μm, t = 7 min)

5. Заключение

Разработан прототип программного комплекса flagmanFoam, предназначенный для моделирования обледенения летательных аппаратов, включающий в себя решатель для расчета нарастания льда в условиях натекания мелких капель (Приложение C к CS-25).

Первые результаты расчетов показали хорошую сходимость (в пределах 12 %) с эталонными данными в различных тестовых задачах: обледенение 2D цилиндра и профиля NACA0012. Размер обледенения во всех расчетных случаях оказывается занижен.

Применяемые принципы позволили добиться высокой скорость вычислений на 1 ядре (время расчета составляет в пределах 1 дня), параллельная масштабируемость программы приближена к стандартному решателю стационарных задач.

Использование неявного представления межфазной поверхности в новом решателе позволило проводить вычисления аккреции льда в условиях размеров нароста сопоставимых или превышающих размер тела.

При этом удалось сохранить модульность и расширяемость функционала физико-математической модели и аппроксимирующего её численного алгоритма, а также их соответствие области применения разрабатываемого вычислительного средства.

В дальнейшем планируется расширять функционал программного комплекса с помощью добавления новых физико-математических моделей, что позволит рассчитывать обледенение в условиях натекания крупных капель, в условиях падающего снега и метели, а также предсказывать нарастание льда других типов.

Список литературы / References

[1] Миллер А.Б., Потапов Ю.Ф. и др. Лабораторная аэрохолодильная установка для исследования процессов обледенения. Ученые записки ЦАГИ, том 47, no. 4, 2016 г., стр. 55-61 / Miller A.B., Potapov Yu.F. et al. Laboratory aero-refrigeration setup for investigation of ice accretion processes. TsAGI Science Journal, vol. 47, no. 4, 2016, pp. 423-432.

[2] Кашеваров А.В., Левченко В.С. и др. К гидротермодинамике обледенения профиля в воздушно-кристаллическом потоке. Журнал технической физики, том 88, вып. 6, 2018 г., стр. 808-814 / Kashevarov A.V., Levchenko V.S. On the hydrothermodynamics of the icing of a wing profile in the air-crystalline flow. Technical Physics, vol. 63, issue 6, 2018, pp. 782-788.

[3] Thomas S.K., Cassoni R.P., MacArthur CD. Aircraft Anti-Icing and De-Icing Techniques and Modeling. Journal of Aircraft, vol. 33, issue 5, 2012, pp. 841–854.

[4] Miller A.B., Potapov Yu.F., Stasenko A.L. Experimental and Theoretical Investigations of Aircraft Icing in the Case of Crystal and Mixed-phase Flow. In Proc. of the 29th Congress of the International Council of the Aeronautical Sciences, 2014, paper 2014-0575.

[5] Bidwell C.S. Collection Efficiency and Ice Accretion Calculations for a Boeing 737-300 Inlet. SAE/AIAA Technical Paper 96-5570, 21p.

[6] CS-25 Large Aeroplanes. Available at: https://www.easa.europa.eu/certification-specifications/cs-25-large-aeroplanes, accessed 25.10.2021.

[7] Xu Yu., Liu X. An immersed boundary method with y+-adaptation wall function for smooth wall shear. International Journal of Numerical Methods in Fluids, vol. 93, issue 6, 2021, pp. 1929-1946.

[8] GitHub-psu-efd/ibwallfunction_OpenFOAM: An immersed boundary method with y+-adaptive wall function for smooth wall shear. Available at: https://github.com/psu-efd/ibwallfunction_OpenFOAM, accessed 25.10.2021.

[9] Myers T.G. Extension to the Messinger Model for Aircraft Icing. AIAA JOURNAL, vol. 39, no. 2, 2001, pp. 211-218.

[10] Dianat M., Skarysz M., Garmory A. A Coupled Level Set and Volume of Fluid method for automotive exterior water management applications. International Journal of Multiphase Flow, vol. 91, 2017, pp. 19-38.

[11] Anderson D.N. Rime-, Mixed- and Glaze-ice Evaluations of Three Scaling Laws. In Proc. of the AIAA 32nd Aerospace Sciences Meeting and Exhibit, 1994, 16 p.

[12] Icing Wind Tunnel Interfacility Comparison Tests. SAE Aerospace information report AIR5666. 2012.

[13] Wright W.B., Rutkowski A. Validation Results for LEWICE 2.0. Technical Report NASA/CR-1999-208690, 1999, 679 p.

Информация об авторах / Information about authors

Кирилл Александрович ВАТУТИН – аспирант, его научные интересы включают задачи волновой динамики в стратифицированных и вращающихся средах, разработка программных средств для задач гидроаэродинамики.

Kirill Alexandrovich VATUTIN is a PhD student, his scientific interests include the problems of wave dynamics in stratified or rotating media, and development of CFD software.

Матвей Викторович КРАПОШИН – руководитель лаборатории СПО ЦМТС в ИСП РАН. Область научных интересов: гидродинамика, численное моделирование, многомасштабное моделирование, программное обеспечение с открытым исходным кодом, сжимаемые течения, двухфазные течения, междисциплинарные модели.

Matvey Viktorovich KRAPOSHIN is a head of the laboratory of OSS DMTS at ISP RAS. Research interests: hydrodynamics, numerical modeling, multiscale modeling, open-source software, compressible flows, two-phase flows, interdisciplinary models.

Максим Александрович КУДРОВ – кандидат технических наук, доцент, ведущий научный сотрудник, заведующий лабораторией информационных технологий и прикладной математики. Научные интересы: разработка комплекса программ для компьютерного моделирования физических процессов.

Maksim Aleksandrovich KUDROV is a Candidate of Technical Sciences, Associate Professor, Leading Researcher, Head of the Laboratory of Information Technologies and Applied Mathematics. Research interests: development of software for computer modeling of physical processes.

Алексей Борисович МИЛЛЕР – кандидат физико-математических наук, начальник отдела отделения исследований аэротермодинамики гиперзвуковых летательных аппаратов и объектов ракетно-космической техники. Научные интересы: физическая механика многофазных потоков, численное моделирование, экспериментальные исследования.

Alexey Borisovich MILLER is a Candidate of Sciences in Physics and Mathematics, Head of the Department for Aerothermodynamics Research of Hypersonic Aircraft and Rocket and Space Facilities. Research interests: physical mechanics of multiphase flows, numerical modeling, experimental research.

Валерия Геннадиевна МЕЛЬНИКОВА – аспирант МГТУ, кафедра «Аэрокосмические системы», научный сотрудник ИСП РАН. Сфера научных интересов: вычислительная гидродинамика, метод контрольного объема, подвижные сетки, лагранжев подход.

Valeriia Gennadievna MELNIKOVA is a PhD student of BMSTU, «Aerospace systems» department, researcher at ISP RAS. Research interests: computational fluid dynamics, finite volume method, dynamic meshes, particles.

Алексей Олегович МОРОЗОВ – инженер. Научные интересы: компьютерное моделирование физических процессов, численные методы в задачах аэрогидродинамики.

Alexey Olegovich MOROZOV is an engineer. Research interests: computer modeling of physical processes, numerical methods in problems of aerohydrodynamics.

Софья Михайловна САУТКИНА – аспирант МГТУ, сотрудник ИСП РАН. Занимается исследованиями в области гидродинамики, прикладной математики и газовой динамики.

Sofya Mikhailovna SAUTKINA is a PhD student of BMSTU, works at ISP RAS. She does research in Fluid Dynamics, Applied Mathematics and Gas Dynamics.

Александр Алексеевич ШЕВЕЛЕВ – аспирант МГТУ, сотрудник ИСП РАН. Область научных интересов: прикладная математика, численное моделирование, параллельные вычисления.

Alexander Alekseevich SHEVELYOV is a PhD student of BMSTU, works at ISP RAS. Research interests: applied mathematics, numerical modeling, parallel computing.

DOI: 10.15514/ISPRAS-2021-33(6)-18

Возникновение контрастных структур для галактического магнитного поля: теоретические оценки и моделирование на видеокартах

[1,2] *Е.А. Михайлов, ORCID: 0000-0002-9747-4039 <ea.mikhajlov@physics.msu.ru>*
[1] *Т.Т. Хасаева, ORCID: 0000-0003-2431-8024 <justanyan@yandex.ru>*
[3] *И.О. Тепляков, ORCID: 0000-0002-2355-1935 <igor.teplyakov@mail.ru>*

[1] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1.*
[2] *Физический институт имени П.Н. Лебедева РАН, 119991, Россия, г. Москва, Ленинский проспект, д. 53*
[3] *Объединенный институт высоких температур РАН, 125421, Россия, Москва, Ижорская ул., д. 13, стр. 2*

Аннотация. Генерация магнитных полей галактик – важная задача как с точки зрения космической магнитной гидродинамики, так и с позиции вычислительной математики (как правило, моделирование полей требует значительных компьютерных ресурсов, часто необходимо использование параллельных вычислений). Процесс эволюции поля описывается с помощью уравнений динамо среднего поля, которые в общем случае являются нелинейными. Они допускают формирование контрастных структур, предсказываемых теорией сингулярных возмущений, описывающей уравнения с малым параметром при старшей производной. С астрономической точки зрения подобные решения обычно связывают со спиральной структурой галактик и с формированием инверсий магнитного поля: в разных частях галактики формируются области с противоположным направлением магнитного поля, разделенные узкими переходными слоями. С вычислительной точки зрения решение полной двумерной задачи является достаточно ресурсоемкой задачей, поэтому оказывается разумным использование параллельных вычислений. Одним из вариантов реализации данного решения выступает платформа OpenCL, позволяющая в несколько раз увеличить производительность процесса. OpenCL является перспективным кроссплатформенным стандартом для разработки приложений, в частности использующих GPU, производительность которых по мере эволюции драйверов стремительно увеличивается. В настоящей работе представлены основные теоретические оценки поведения магнитного поля, которые в дальнейшем подтверждаются и уточняются в ходе компьютерного моделирования на видеокартах. Показано, что механизм возникновения переходных слоев в радиальном и азимутальном направлениях описывается принципиально различными механизмами. В то время как радиальные инверсии магнитного поля оказываются достаточно устойчивыми, все азимутальные структуры быстро размываются за счет характера течений межзвездного газа. Это означает также практическую нереализуемость возникновения неосесимметричных распределений магнитного поля.

Ключевые слова: магнитная гидродинамика; параллельные вычисления; магнитные поля галактик; контрастные структуры

Для цитирования: Михайлов Е.А., Хасаева Т.Т., Тепляков И.О. Возникновение контрастных структур для галактического магнитного поля: теоретические оценки и моделирование на видеокартах. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 253-264. DOI: 10.15514/ISPRAS–2021–33(6)–18

Благодарности. Работа выполнена с использованием оборудования Центра коллективного пользования сверхпроизводительными вычислительными ресурсами МГУ имени М.В. Ломоносова.

The emergence of contrast structures for galactic magnetic field: theoretical estimates and modeling on GPU

[1,2] *E.A. Mikhailov, ORCID: 0000-0002-9747-4039 <ea.mikhajlov@physics.msu.ru>*
[1] *T.T. Khasaeva, ORCID: 0000-0003-2431-8024 <justanyan@yandex.ru>*
[3] *I.O.Teplyalov, ORCID: 0000-0002-2355-1935 <igor.teplyakov@mail.ru>*

[1] *M.V. Lomonosov Moscow State University,*
Moscow,119991, Russia.
[2] *P.N. Lebedev Russian Academy of science Physical Institute,*
Moscow, 119991, Russia.
[3] *Russian Academy of science Joint Institute of High Temperatures,*
Moscow, 125421, Russia.

Abstract. Magnetic field generation in galaxies turns out to be a significant problem both for cosmic magnetohydrodynamics and mathematical physics. It is based on dynamo mechanism characterising the transition between the energy of medium turbulent motions and the magnetic field energy. The evolution of the field is described with the help of mean field dynamo equations. For galaxies the solutions are commonly found using so-called "no-z" approximation, while the half-thickness of the galactic disc is considered negligible. In nonlinear case mentioned equations admit contrast structure formation, predicted by the singular perturbation theory, describing equations with small parameter at the elder derivative. From astronomical point of view some authors tend to connect such solutions with the spiral structure of the galaxies and the formation of magnetic field reversals (when in different parts of galaxy there are regions with opposite directions of magnetic, divided by a thin transition layer). From numerical point of view finding the solution of two-dimensional system of equations requires large computational resources, for this reason using GPU and parallel calculations turns out to be reasonable. One of the implementation methods is calculating using OpenCL, which allows one to increase the process efficiency several times. OpenCL is a perspective crossplatform standard for development of applications, particularly involving GPU, the efficiency of which is rapidly increasing as the drivers evolve. The present work presents basic theoretical assessments of magnetic field behaviour, which are further confirmed and clarified during the computations. It is shown that the formation of the transition layers is described by fundamentally different mechanisms in radial and azimuthal directions. While radial reversals of the field turn out to be rather stable, all of the azimuthal structures are rapidly blurred due to the nature of the interstellar medium motions. That also indicates the practical impossibility of non-axisymmetric distributions of the field.

Keywords: magnetohydrodynamics; parallel calculations; galactic magnetic field; contrast structures

For citation: Mikhailov E.A., Khasaeva T.T., Teplyakov I.O. The emergence of contrast structures for galactic magnetic field: theoretical estimates and modeling on GPU. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 253-264 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–18

Acknowledgements. The work was performed using the equipment of the Center for Collective Use of Supercomputing Resources of Lomonosov Moscow State University.

1. Введение

Генерация магнитных полей галактик представляет собой одну из классических задач теории динамо [1]. В свою очередь, теория динамо занимает важное место в космической магнитной гидродинамике, объясняя происхождение магнитных полей большого числа астрофизических объектов [2, 3]. Также нельзя не отметить, что большое количество задач, возникающих при обсуждении генерации космических магнитных полей представляют исключительную важность с точки зрения математической физики, являя собой важные примеры реализации тех или иных теоретических представлений [4, 5, 6, 7].

В случае исследования регулярных структур магнитных полей важно отметить, что уравнения галактического динамо являются результатом усреднения, которое ведется по областям, размеры которых превышают типичные масштабы турбулентности для

межзвездного газа [1, 3, 8]. Это приводит к появлению в уравнениях для магнитного поля дополнительных слагаемых по сравнению с классическими уравнениями магнитной гидродинамики. Ключевую роль играют альфа-эффект и дифференциальное вращение [9, 10].

Альфа-эффект связан с наличием так называемой спиральности турбулентных движений [1]. В случае галактик (а также большого числа других астрофизических объектов) завихренность течений имеет ненулевую среднюю проекцию на направление скорости. Кроме того, особую роль играют также противоположные знаки завихренности в разных полушариях – так называемая зеркальная асимметрия альфа-эффекта [9, 10]. Все это приводит к тому, что вмороженное магнитное поле интенсивно вращается за счет подобных течений. Также важно отметить, что по мере роста магнитного поля энергия турбулентных движений сокращается, что приводит к насыщению его роста. В таком случае уравнения, описывающие генерацию магнитного поля, становятся нелинейными [10].

Дифференциальное вращение характеризует тот факт, что большинство галактик вращаются по закону, далекому от твердотельного. Это означает, что угловая скорость уменьшается по мере удаления от оси вращения, и приводит как к повороту магнитного поля, так и к его усилению [2].

Как правило, уравнения для генерации магнитного поля удобнее решать в рамках различных приближений, учитывающих геометрические параметры исследуемого объекта. Так, для галактик наибольшую популярность получило так называемое планарное приближение [11]. Оно принимает во внимание тот факт, что большинство галактических дисков – достаточно тонкие, что позволяет несколько упростить решаемую задачу. Уравнения планарного приближения представляют собой систему нелинейных параболических уравнений с малым параметром при операторе Лапласа.

Подобная задача представляет собой почти классическую задачу о возникновении контрастных структур. Они предсказываются в рамках теории сингулярных возмущений, хорошо известной в математической физике [12, 13]. В таком случае оказывается возможной генерация магнитных полей, которые имеют противоположные направления в разных частях галактики, разделенных узкими переходными слоями.

Данные решения вызывают большой интерес также и с точки зрения астрономии [14, 15, 16, 17, 18, 19, 20]. В случае Млечного Пути хорошо известно, что его магнитное поле демонстрирует двукратную смену направления: она происходит на расстоянии около 5 кпк и 7 кпк от оси вращения [1]. Таким образом, области с магнитным полем одного и того же направления образуют концентрические области круговой формы. В настоящий момент существуют серьезные основания предполагать возможность существования подобных структур и в других галактических объектах, для которых имеются данные о фарадеевском вращении плоскости поляризации для проходящих через них радиоволн. Ряд работ также предполагал возможность существования неосесимметричных контрастных структур, ассоциируемых со спиральной структурой галактик. Вместе с тем, согласно современным астрономическим наблюдениям возможность их возникновения является сомнительной [6].

Отметим, что построение достаточно точной модели возникновения контрастных структур в галактике до сих пор является затруднительным. Большинство имеющихся теоретических представлений связаны либо с единственным эволюционным уравнением, либо с парой уравнений, в которых присутствуют принципиально разные временные масштабы. В то же время, в случае галактического поля мы имеем систему из как минимум двух уравнений магнитной гидродинамики, в каждом из которых присутствует один и тот же коэффициент турбулентной диффузии при лапласиане. По этой причине ответ на вопрос о том, как именно они устроены и как происходит их генерация, может быть дан лишь с численных позиций [21, 22]. Задачи, связанные с генерацией магнитных полей, оказываются весьма ресурсоемкими (особенно в тех случаях, когда необходимо рассмотреть влияние различных случайных эффектов в начальных данных и провести серию вычислений). Поэтому

практически неизбежным является использование параллельных вычислений. В настоящий момент один из самых распространенных подходов связан с использованием вычислений на видеокартах [23, 24, 25]. Он позволяет заметно ускорить процесс расчета, добившись значимых результатов за короткие временные промежутки.

В настоящей работе представлены основные теоретические модели для генерации магнитных полей и возникновения инверсий. Исследован вопрос о том, насколько возможным может быть генерация полей той или иной симметрии, а также как они могут быть связаны с параметрами, характерными для конкретных галактических объектов. После этого проводится численное исследование соответствующего процесса с применением параллельных вычислений с использованием графических процессоров, которое позволяет окончательно установить, является ли тот или иной результат возможным. Исследуется вопрос о движении переходных слоев между областями с противоположным магнитным полем, дан ответ на вопрос о том, насколько подобное движение может разрушить сформировавшиеся контрастные структуры магнитного поля. Наконец, в работе даются оценки того, как полученные результаты могут быть ассоциированы с теми или иными астрономическими наблюдениями.

2. Эволюция контрастных структур

Генерация магнитного поля в галактике описывается с помощью уравнения динамо среднего поля. Оно исходит из того, что магнитное поле является результатом усреднения по масштабам, соответствующим размерам турбулентных ячеек:

$$\mathbf{B} = \langle \mathbf{H} \rangle.$$

Тогда уравнения магнитной гидродинамики сводятся к виду [1]:

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\alpha \mathbf{B}) + \nabla \times [\mathbf{V} \times \mathbf{B}] + \eta \Delta \mathbf{B};$$

где введен коэффициент, отвечающий за альфа-эффект [1]:

$$\alpha = -\frac{\tau}{3} \langle \boldsymbol{\upsilon} \cdot \nabla \times \boldsymbol{\upsilon} \rangle;$$

где τ - корреляционное время, $\boldsymbol{\upsilon}$ – скорость турбулентных движений.

Отметим, что по мере роста магнитного поля скорости турбулентных движений будут замедляться. Это может быть учтено с помощью следующей модели:

$$\alpha(r, \mathbf{B}) = \alpha_0(r) \left(1 - \frac{B^2}{B^{*2}} \right);$$

где $\alpha_0(r)$ связан с пространственной структурой течений, а B^* – так называемое поле равнораспределения, определяемое по закону [10]:

$$B^* = 2\upsilon_0 \sqrt{\pi \rho};$$

где ρ - плотность среды, а υ_0 – начальное значение скорости турбулентных движений межзвездного газа.

Также данная система уравнений содержит крупномасштабную скорость движений, которая также является результатом усреднения и может быть ассоциирована с вращением галактики [11]:

$$\mathbf{V} = \langle \boldsymbol{\upsilon} \rangle = r\Omega \mathbf{e}_\varphi.$$

В свою очередь, η представляет собой так называемый коэффициент турбулентной диффузии.

Если воспользоваться планарным приближением, использующим тот факт, что галактический диск достаточно тонкий, и применить ряд стандартных приближений, мы можем получить следующую систему уравнений [11, 21]:

$$\frac{\partial B_r}{\partial t} = -\frac{\alpha(r)}{h} B_\varphi \left(1 - \frac{B_r^2 + B_\varphi^2}{B^{*2}}\right) - \Omega \frac{\partial B_r}{\partial \varphi} - \frac{\pi^2}{4h^2} B_r + \eta \left\{ \frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial}{\partial r}\left(rB_r\right)\right) + \frac{1}{r^2}\frac{\partial^2 B_r}{\partial \varphi^2} - \frac{2}{r^2}\frac{\partial B_\varphi}{\partial \varphi} \right\};$$

$$\frac{\partial B_\varphi}{\partial t} = r\frac{d\Omega}{dr} B_r - \Omega \frac{\partial B_\varphi}{\partial \varphi} - \eta \frac{\pi^2}{4h^2} B_\varphi + \eta \left\{ \frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial}{\partial r}\left(rB_\varphi\right)\right) + \frac{1}{r^2}\frac{\partial^2 B_\varphi}{\partial \varphi^2} + \frac{2}{r^2}\frac{\partial B_r}{\partial \varphi} \right\}.$$

Отметим, что скорость роста магнитного поля обусловлена совместным действием альфа-эффекта и дифференциального вращения. Скорость роста магнитного поля описывается с помощью выражения [22]:

$$\gamma(r) = \sqrt{\frac{\alpha(r)}{h} r \frac{d\Omega}{dr}} - \eta \frac{\pi^2}{4h^2}$$

Кроме того, анализ решений уравнений планарного приближения показывает, что основную роль в эволюции поля играет азимутальная компонента. Таким образом, можно считать, что $B_r << B_\varphi$ (при этом важно отметить, что хотя радиальное магнитное поле и является достаточно малым, нельзя пренебрегать им полностью).

Эволюция основной компоненты магнитного поля может описываться при помощи следующего качественного уравнения (опыт моделирования показывает, что оно принципиально правильно описывает большинство происходящих процессов) [22]:

$$\frac{\partial B_\varphi}{\partial t} = \gamma(r) B_\varphi \left(1 - \frac{B_\varphi^2}{B^{*2}}\right) - \Omega \frac{\partial B_\varphi}{\partial \varphi} + \eta \left\{ \frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial}{\partial r}\left(rB_\varphi\right)\right) + \frac{1}{r^2}\frac{\partial^2 B_\varphi}{\partial \varphi^2} \right\}.$$

Особый интерес представляют решения, соответствующие так называемым контрастным структурам, когда в разных частях объекта может наблюдаться устойчивые магнитные поля противоположных направлений. В Млечном Пути существуют серьезные основания предполагать наличие резкой смены знака магнитного поля в радиальном направлении на расстоянии от центра, соответствующем примерно 5 кпк и 7 кпк [20, 21]. Кроме того, в ряде работ предполагается, что контрастные структуры могут в определенном смысле повторять спиральную структуру галактики.

Возникновение контрастной структуры – принципиально нелинейный процесс, который описывается в рамках так называемой теории сингулярных возмущений в математической физике [21, 22]. Обсудим принципиальную возможность возникновения «радиальных» и «азимутальных» инверсий магнитного поля.

В том случае, если мы исследуем «перескок» магнитного поля в радиальном направлении, ключевую роль будут играть производные магнитного поля, соответствующие угловой переменной. Кроме того, будем предполагать, что мы исследуем область на достаточно большом расстоянии от центра объекта, поэтому оператор Лапласа приближенно будет выглядеть следующим образом:

$$\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial}{\partial r}\left(rB_\varphi\right)\right) + \frac{1}{r^2}\frac{\partial^2 B_\varphi}{\partial \varphi^2} \cong \frac{\partial^2 B_\varphi}{\partial r^2} + \frac{1}{r}\frac{\partial B_\varphi}{\partial r}.$$

Как и принято в асимптотической теории контрастных структур, будем искать решение в автомодельном виде [13]:

$$B(r,t) = U(\xi),$$

где $\xi = \frac{r - r^*}{\sqrt{\eta}}$, r^* - расстояние, соответствующее локализации переходного слоя (возможно, очень медленно зависящее от времени).

Тогда для данной функции $U(\xi)$ с учетом того, что $|r - r^*| << r^*$, пропорциональными степеням мы получим следующее уравнение с граничными условиями:

$$-\frac{1}{\sqrt{\eta}}\frac{dr^*}{dt}U' = \gamma U\left(1 - \frac{U^2}{B^{*2}}\right) + U'' + \frac{\sqrt{\eta}}{r^*}U'$$

$$\lim_{\xi \to \pm\infty} U = \pm B^*$$

Предполагая, что скорость $\frac{dr^*}{dt}$ пропорциональна η, и оставляя только слагаемые порядка η^0, мы можем получить следующее уравнение для приближения для функции U:

$$U'' + \gamma(r^*)U\left(1 - \frac{U^2}{B^{*2}}\right) = 0.$$

Его решение выглядит так [7]:

$$U(\xi) = B^* \tanh\left(\xi\sqrt{\frac{\gamma}{2}}\right)$$

Исследуем теперь возможность движения структуры магнитного поля. Для этого учтем что $\gamma \cong \gamma(r^*) + \xi\sqrt{\eta}\gamma'(r^*)$. Тогда, учитывая в уравнении для поля слагаемые порядка $\sqrt{\eta}$, мы получим соотношение:

$$-\frac{1}{\sqrt{\eta}}\frac{dr^*}{dt}U' = \gamma'(r^*)U\left(1 - \frac{U^2}{B^{*2}}\right) + \frac{\sqrt{\eta}}{r^*}U'$$

Перепишем уравнение в форме:

$$\left(\frac{dr^*}{dt} + \frac{\eta}{r^*}\right)U' = -\gamma'(r^*)\eta U\left(1 - \frac{U^2}{B^{*2}}\right);$$

Домножив обе части на U' и проинтегрировав их по всей числовой оси, мы получим [13]:

$$\frac{dr^*}{dt} + \frac{\eta}{r^*} = -\eta\gamma'(r^*)\frac{\int_{-1}^{+1} U\left(1 - \frac{U^2}{B^{*2}}\right)dU}{\int_{-\infty}^{+\infty}(U')^2 d\xi}.$$

Подставляя найденное выражение для функции U, мы получим [22]:

$$\frac{dr^*}{dt} = -\eta\left(\frac{1}{r^*} + \frac{\gamma'(r^*)}{2\gamma(r^*)}\right)$$

Отметим, что учитывая малую величину вязкости, скорость перемещения будет достаточно малой. Это позволит существовать устойчивым контрастным структурам. Кроме того,

характерное время возникновения контрастной структуры $\frac{1}{\gamma}$ будет меньше, чем время ее перемещения и разрушения:

$$\frac{1}{\gamma} << \frac{r^*}{\frac{dr^*}{dt}}.$$

Исследуем теперь вопрос о возможности генерации структур в азимутальном направлении. Это означает, что от радиальной координаты поле будет зависеть слабо, а оператор Лапласа - выглядеть так:

$$\left\{\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial}{\partial r}\left(rB_\varphi\right)\right)+\frac{1}{r^2}\frac{\partial^2 B_\varphi}{\partial\varphi^2}\right\}\cong\frac{1}{r^2}\frac{\partial^2 B_\varphi}{\partial\varphi^2}.$$

В таком случае уравнение для магнитного поля представляется в форме:

$$\frac{\partial B_\varphi}{\partial t}=\gamma(r)B_\varphi\left(1-\frac{B_\varphi^2}{B^{*2}}\right)-\Omega\frac{\partial B_\varphi}{\partial\varphi}+\eta\frac{1}{r^2}\frac{\partial^2 B_\varphi}{\partial\varphi^2}.$$

В таком случае производная магнитного поля по азимутальному углу не может быть пренебрежимо малой по сравнению с остальными слагаемыми. Поэтому необходимо искать поле в следующей форме:

$$B_\varphi = B_\varphi(\varphi-\Omega t).$$

Это приводит уравнение к виду:

$$\gamma B_\varphi\left(1-\frac{B_\varphi^2}{B^{*2}}\right)+\eta\frac{B_\varphi''}{r^2}=0.$$

Его решение представляется в виде:

$$B_\varphi=\tanh\left(\frac{1}{r}\sqrt{\frac{\gamma}{2\eta}}(\varphi-\Omega t)\right)$$

Данная структура будет двигаться с азимутальной скоростью Ω. Типичное время движения этой структуры $\frac{1}{\Omega}$ будет вполне сопоставимо со временем генерации поля $\frac{1}{\gamma}$:

$$\frac{1}{\gamma}\sim\frac{1}{\Omega}.$$

Кроме того, возникновению контрастных структур азимутального типа мешает то, что скорость вращения отличается для различных расстояний от центра. Отметим, что при малых значениях угловой скорости не будет выполнено ключевое условие генерации поля – наличие дифференциального вращения.

Все это говорит о том, что генерация контрастных структур азимутального типа представляется маловероятной.

3. Решение задачи на видеокарте

Решение подобных задач требует высоких вычислительных мощностей ввиду необходимости перебора ряда решений. Характерное время формирования переходного слоя можно определить при помощи численного эксперимента, что подразумевает проверку работы программы для разных характерных времен расчета. Для решения данной проблемы

в задаче использовались параллельные методы вычисления на видеокартах [24]. Данный подход позволяет сократить время работы программы в десятки раз. Также это позволяет рассмотреть ряд различных начальных условий с переходными слоями и сравнить поведение магнитного поля с основными теоретическими предположениями.

Из численных соображений будет удобно использовать декартовую прямоугольную сетку по пространству [24]. Для этого необходимо ввести соответствующую замену:

$$x_i = r\cos\varphi$$
$$y_j = r\sin\varphi$$

Здесь x_i и y_j – координаты узла сетки, соответствующего индексу [i,j]. Также из практических соображений на границе галактического диска и за его пределами поле обращается в нуль:

$$B_{r,\varphi}|_{r\geq R} = 0.$$

В действительности процессы, преобладающие в центральной части галактики, не могут быть описаны в рамках теории динамо, а также в ряде случаев обладают специфическими особенностями и не терпят обобщения. Таким образом, из области вычисления имеет смысл исключить зону, соответствующую центральной области галактики:

$$B_{r,\varphi}|_{r\leq r_{min}} = 0\ ,$$

где r_{min} – радиус вышеуказанной центральной области.

Введем значения пространственных переменных в узлах сетки по правилу: $x^i = -R + i\cdot\Delta x$ и $y^j = -R + j\cdot\Delta y$, где Δx и Δy – шаги сетки, R – характерный размер области, а i и j – индексы, принимающие значения в диапазоне $0 \leq i \leq N-1, 0 \leq j \leq M-1, \Delta x = \frac{2R}{N-1},\ \Delta y = \frac{2R}{M-1}$. Поставим в соответствие величинам компонент магнитного поля в моменты времени t и $t+\Delta t$ следующие сеточные значения:

$$B_r(t) \to {Br_0}^{i,j};$$
$$B_\varphi(t) \to {Bf_0}^{i,j};$$
$$B_r(t+\Delta t) \to Br^{i,j};$$
$$B_\varphi(t+\Delta t) \to Bf^{i,j}.$$

Сеточные значения функции на следующем шаге можно вычислить так (подобный подход обеспечивает первый порядок по времени и второй по пространству):

$$Br^{i,j} = \left(-\alpha^{i,j}\pi\cdot {Bf_0}^{i,j} - \frac{\eta\pi^2 {Br_0}^{i,j}}{4h^2}\right)\cdot\Delta t\ +\ {Br_0}^{i,j}\ +$$
$$+\ \Omega^{i,j}\cdot\left(-y^j\cdot\frac{{Br_0}^{(i+1),j} - {Br_0}^{(i-1),j}}{2\cdot\Delta x}\ +\ x^i\cdot\frac{{Br_0}^{i,(j+1)} - {Br_0}^{i,(j-1)}}{2\cdot\Delta y}\right)\cdot\Delta t\ +$$
$$+\eta\cdot\left(\frac{Br_0^{(i+1),j} - 2\cdot Br_0^{i,j} + Br_0^{(i-1),j}}{\Delta x^2} + \frac{Br_0^{i,(j+1)} - 2\cdot Br_0^{i,j}\ +\ Br_0^{i,(j-1)}}{\Delta y^2} - \frac{{Br_0}^{i,j}}{(r^{ij})^2} -\right.$$
$$\left. -\left(-y^j\cdot\frac{Bf_0^{(i+1),j} - Bf_0^{(i-1),j}}{2\cdot\Delta x}\ +\ x^i\cdot\frac{Bf_0^{i,(j+1)} - Bf_0^{i,(j-1)}}{2\cdot\Delta y}\right)\cdot\frac{2}{(r^{i,j})^2}\right)\cdot\Delta t$$

$$Bf^{i,j} = -\frac{\Omega_0\cdot r^{i,j}}{\sqrt{\left(1+\frac{(r^{,ij})^2}{r_0^2}\right)^3}}\cdot {Br_0}^{i,j}\cdot\Delta t\ -\frac{\eta\pi^2 Bf_0^{i,j}}{4h^2}\cdot\Delta t\ +\ Bf_0^{i,j}\ +$$
$$+\ \Omega^{i,j}\cdot\left(-y^j\cdot\frac{Bf_0^{(i+1),j} - Bf_0^{(i-1),j}}{2\cdot\Delta x} + x^i\cdot\frac{{Bf_0}^{i,(j+1)} - {Bf_0}^{i,(j-1)}}{2\cdot\Delta y}\right)\cdot\Delta t\ +$$
$$+\eta\cdot\left(\frac{Bf_0^{(i+1),j} - 2\cdot Bf_0^{i,j} + Bf_0^{(i-1),j}}{\Delta x^2} + \frac{Bf_0^{i,(j+1)} - 2\cdot Bf_0^{i,j}\ +\ Bf_0^{i,(j-1)}}{\Delta y^2} - \frac{Bf_0^{i,j}}{(r^{i,j})^2}\ +\right.$$
$$\left. +\ \frac{2}{(r^{i,j})^2}\cdot\left(-y^j\cdot\frac{Br_0^{(i+1),j} - Br_0^{(i-1),j}}{2\cdot\Delta x} + x^i\cdot\frac{Br_0^{i,(j+1)} - Br_0^{i,(j-1)}}{2\cdot\Delta y}\right)\right)\cdot\Delta t$$

Выше использованы следующие значения коэффициентов:

$$r^{i,j} = \sqrt{(x^i)^2 + (y^j)^2};$$
$$\Omega^{i,j} = \Omega(r^{i,j});$$
$$\alpha^{i,j} = \frac{\alpha_0}{h}\frac{r_0}{r^{i,j}}\left(1 - \frac{\left(Br_0{}^{i,j}\right)^2 + \left(Bf_0{}^{i,j}\right)^2}{B_0{}^2}\right).$$

Граничные условия будут соответствовать обращению компонент поля в нуль при $r^{i,j} \geq R$ и при $r^{i,j} \leq r_{min}$.

Применяемая нами численная схема хорошо распараллеливается, хотя и обладает и рядом недостатков, в том числе условной сходимостью. Тем не менее, вышеупомянутая проблема успешно решается благодаря использованию вычислительных возможностей GPU. Использование видеокарт позволяет увеличить вычислительную мощность в несколько десятков раз.

С целью минимизировать непредвиденные сбои на видеокарте при расчете магнитного поля в каждой точке галактики вместо двумерных массивов используются одномерные Br_n и Bf_n. Таким образом, номер каждого элемента в таком одномерном массиве будет записываться через соответствующие индексы элемента из двумерного массива:

$$n = i + N \cdot j$$

Теперь аналитически оценим выигрыш в вычислительной мощности (K) в случае использования видеокарты. Для сравнения рассматриваются видеокарта GPU GTX 660 и процессор Intel i7. Верхнюю априорную оценку даваемой выгоды (важно учитывать, что она является оценкой «с большим запасом») можно провести, воспользовавшись формулой:

$$K = \frac{1024 \cdot \omega_{GPU}}{\omega_{CPU}},$$

где ω_{GPU} и ω_{CPU} - тактовая частота процессора на видеокарте и на ЦП соответственно. Согласно оценке, $K \approx 87$. Отметим также, что не только тактовая частота играет существенную роль в оценке преимуществ параллельного подхода. Существенную роль также играют другие нюансы, связанные с более медленным выполнение ряда операций на видеокарте, поэтому на деле данная оценка актуальна скорее в смысле порядка величины, а более точно величину *K* необходимо вычислять опытным путем.

Время работы программы на некоторых из использованных нами видеокарт представлены в Табл. 1. Данные получены для значений t=8 млрд. лет.

Табл. 1. Время расчета задачи на GPU и CPU

Тип процессора	**CPU Intel i7-**	**GPU GTX 660**	**GPU Titan Black**
Время расчета (сетка 200x200), с	8357	297	103
Время расчета (сетка 400x400), с	31976	895	174
Время расчета (сетка 800x800), с	110214	3486	795

Для начала рассмотрим начальные условия неоднородные по радиальному направлению:

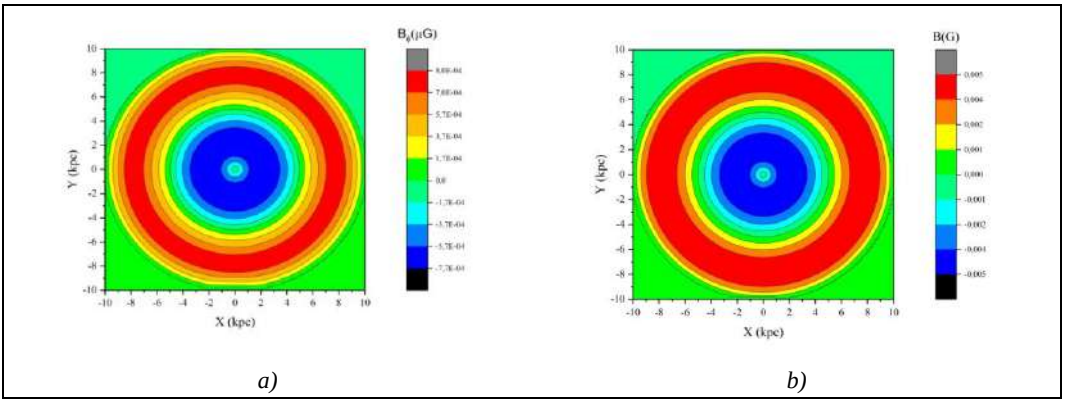
$$\mathrm{B}_\varphi = B_0 \frac{r}{10}\left(1 - \frac{r}{10}\right)\left(\frac{r}{10} - \frac{1}{2}\right)$$
$$\mathrm{B}_r = 0$$

Предполагается, что инверсии по радиальному направлению не вырождаются со временем по мере вращения галактического диска. Результаты работы программы представлены на рис. 1.

Для полноты картины рассмотрим такое начальное поле, которое будет содержать в себе как азимутальные, так и радиальные неоднородности:

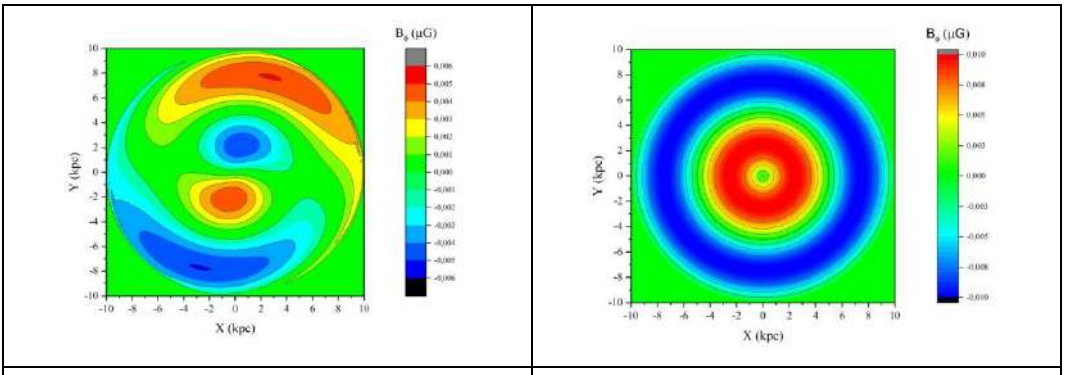
$$\mathrm{B}_\varphi = B_0 \frac{r}{10}\left(1 - \frac{r}{10}\right)\left(\frac{r}{10} - \frac{1}{2}\right) sin\varphi$$
$$\mathrm{B}_r = B_0\left(-\frac{r^2}{300} + \frac{r^3}{4000} + \frac{r}{40} - \frac{r^2}{600}\right) cos\varphi$$

Результат эволюции поля, представленный для 12 миллиардов лет, указывает на соответствие наших теоретических предположений действительности: азимутальные неоднородности не являются устойчивыми и в конечном счете вырождаются (рис. 2).



a) *b)*

Рис. 1. Начальное распределение (a) и результат эволюции поля (b) с радиальными инверсиями
Fig. 1. Initial distribution (a) and the result of evolution (b) of the field with radial reversals



a) *b)*

Рис. 2. Начальное распределение (a) и результат эволюции поля (b) с азимутальными и радиальными инверсиями
Fig. 2. Initial distribution (a) and the result of evolution (b) of the field with both radial and angular reversals

4. Выводы

Результаты расчетов галактического магнитного поля подтвердили изначальные предположения об азимутальных и радиальных инверсиях поля. Согласно графикам, любая азимутальная неоднородность склонна к сглаживанию со временем. При этом численный эксперимент подтвердил устойчивость радиальных инверсий, что отчетливо видно на графике с результатами эволюции поля, в котором присутствуют радиальные неоднородности. Таким образом, можно сделать вывод о неустойчивости азимутальных и устойчивости радиальных инверсий, на что указывали и более ранние работы [6][20].

Список литературы / References

[1] Krause F., Rädler K.-H. Mean-Field Magnetohydrodynamics and Dynamo. Pergamon, 1980, 271 p.
[2] Соколов Д.Д. Проблемы магнитного динамо. Успехи физических наук, том 185, no. 6, 2015 г., стр. 643-648 / Sokoloff D.D. Problems of magnetic dynamo. Physics-Uspekhi, vol. 58, no. 6, pp/ 601-605.
[3] Степанов Р.А., Фрик П.Г., Соколов Д.Д. Сопряжение уравнений динамо средних полей и каскадной модели турбулентности в проблеме галактического динамо. Вычислительная механика сплошных

сред, том. 1, no. 4, 2008 г., стр. 97-108 / Stepanov R., Frick P., Sokoloff D. Coupling of mean-field equation and shell model of turbulence in the context of galactic dynamo problem. Computational Mechanics of Continuous Media, vol. 1, no. 4, 2008, pp. 97-108 (in Russian).

[4] Михайлов Е.А. Спектральное разложение решения задачи о генерации магнитных полей галактик в планарном приближении. Вестник Московского университета. Серия 3. Физика. Астрономия, no. 5, 2020 г., стр. 39-44 / Mikhailov E.A. The spectral decomposition of the solution of the problem of generating galactic magnetic fields in the no-z approximation. Moscow University Physics Bulletin, vol. 75, no. 5, pp. 420-426.

[5] Mikhailov E.A. Symmetry of the magnetic fields in galactic dynamo and the material arms. Magnetohydrodynamics, vol. 56, no. 4, 2020, pp. 303-315.

[6] Mikhailov E., Boneva D., Pashentseva M. No-z model for magnetic fields of different astrophysical objects and stability of the solutions. Data, vol. 6, no. 1, article no. 4.

[7] Михайлов Е.А. Задачи с малым параметров и распространение фронтов в теории галактического динамо. Вестник Московского университета. Сер.3. Физика. Астрономия, no. 2, 2015 г., стр. 27-31 / Mikhailov E.A. Problems with a small parameter and propagation of fronts in the galactic dynamo theory. Moscow University Physics Bulletin, vol. 70, no. 2, 2015, pp. 101-106.

[8] Moss D., Stepanov R. et al. Multiscale magnetic fields in spiral galaxies: evolution and reversals. Astronomy and Astrophysics, vol. 537, 2012, article no. 68.

[9] Beck R., Brandenburg A. et al. Galactic Magnetism: Recent Developments and Perspectives. Annual Review of Astronomy and Astrophysics, vol. 34, 1996, pp.155-206.

[10] Arshakian T.G., Beck R. et al. Evolution of magnetic fields in galaxies and future observational tests with the Square Kilometre Array. Astronomy and Astrophysics, vol.494, no. 1, 2009, pp. 21-32.

[11] Moss D. On the generation of bisymmetric magnetic field structures in spiral galaxies by tidal interactions. Monthly Notices of Royal Astronomical Society, vol. 275, issue 1, 1995, pp. 191-194.

[12] Нефедов, Н.Н. Общая схема асимптотического исследования устойчивых контрастных структур. Нелинейная динамика, том 6, no. 1, 2010 г., стр. 181-186 / Nefyodov N.N. General scheme of asymptotic investigation of stable contrast structures. Nonlinear Dynamics, vol. 6, no. 1, 2010, pp. 181-186 (in Russian).

[13] Божевольнов Ю. В., Нефедов Н. Н. Движение фронта в параболической задаче реакция — диффузия // Журнал вычислительной математики и математической физики. — 2010. — Т. 50, № 2. — С. 276–285.

[14] Morris D., Berge G. Direction of the galactic magnetic field in the vicinity of the Sun. Astrophysical Journalб vol. 139, 1964, pp. 1388–1392.

[15] Manchester R. Pulsar rotation and dispersion measures and the galactic magnetic field. Astrophysical Journal, vol. 172, 1973, pp. 43-52.

[16] Beck R. Magnetic fields in spiral galaxies. Astronomy and Astrophysics Review, vol. 24, 2015, article no. 4.

[17] Oppermann, N., Junklewitz. et al. An improved map of the galactic Faraday sky. Astronomy and Astrophysics, vol. 542, 2012, article no. A93.

[18] C. Horrelou, R. Beck et al. Faraday effects in the spiral galaxy M51. Astronomy and Astrophysics. vol. 265, 1992, pp. 417-428.

[19] Frick, P., Stepanov, R. et al. Magnetic and gaseous spiral arms in M83. Astronomy and Astrophysics. vol. 585, 2016, article no. A21.

[20] Van Eck C.L., Brown J.C. et al. Modeling the magnetic field in the galactic disk using new rotation measure observations from the Very Large Array. Astrophysical Journal, vol. 728, no. 2, 2011, article no 97.

[21] Андреасян Р.Р., Михайлов, Е.А., Андреасян А.Р. Структура и особенности формирования инверсий галактического магнитного поля. Астрономический журнал, том 97, no. 3, 2020 г., стр. 179-189 / Andreasyan R.R., Mikhailov E.A., Andreasyan H.R. Structure and features of the galactic magnetic-field reversals formation. Astronomy Reports, vol. 64, no. 3, 2020, pp. 189-198.

[22] Mikhailov E., Khasaeva T. Evolution of the magnetic field reversals in galaxies. Bulgarian Astronomical Journal, vol.31, 2019, pp. 39-50.

[23] Ивочкин Ю.П., Виноградов Д.А., Тепляков И.О. Численный расчет магнитного поля с использованием технологии CUDA применительно к моделированию электровихревых течений. Математическое и программное обеспечение систем в промышленной и социальной сферах, no. 2, 2015 г, стр. 13-18 / Ivochkin Y.P., Vinogradov D.A., Teplyakov I.O. Numerical calculation of magnetic

field with CUDA technology applied to flow modelling electric vortex flows. Software of Systems in the Industrial and Social Fields, no. 2, 2015, pp. 13-18 (in Russian).
[24] Munshi, A., Gaster B.R. et al. OpenCL Programming Guide. Addison-Wesley Professional, 2011, 646 p.
[25] Kalling R.C., Evans T.E. et al. Accelerating the numerical simulation of magnetic field lines in tokamaks using the GPU. Fusion Engineering and Design, vol. 86, no. 4-5, 2011, pp. 399-406.

Информация об авторах / Information about authors

Евгений Александрович МИХАЙЛОВ – кандидат физико-математических наук, старший научный сотрудник. Сфера научных интересов: магнитная гидродинамика, теория динамо, галактический магнетизм, уравнения со случайными коэффициентами, распространение фронтов в нелинейных задачах.

Evgeni Aleksandrovich MIKHAILOV – Candidate of physical and mathematical sciences, senior researcher. Research interests: magnetohydrodynamics, dynamo theory, galactic magnetism, equations with random coefficients, distribution of transition layers in nonlinear problems.

Татьяна Тимуровна ХАСАЕВА – студентка второго курса магистратуры кафедры математики физического факультета. Научные интересы: магнитная гидродинамика, космический магнетизм, вычисления на видеокартах.

Tatiana Timurovna KHASAEVA – second-year Master's student of the department of mathematics of faculty of physics. Research interests: magnetohydrodynamics, cosmic magnetism, modeling on GPU.

Игорь Олегович ТЕПЛЯКОВ – кандидат технических наук, старший научный сотрудник. Научные интересы: магнитная гидродинамика, тепломассообмен, численные методы.

Igor Olegovich TEPLYAKOV – Candidate of technical sciences, senior researcher. Research interests: magnetohydrodynamics, heat and mass transfer, numerical methods.

DOI: 10.15514/ISPRAS-2021-33(6)-19

Моделирование операционных, программных и технических систем в проектах РФФИ

[1,2] *Е.М.Лаврищева, ORCID: 0000-0002-1160-1077 <lavr@ispras.ru>*
[1,3,4] *А.К. Петренко, ORCID: 0000-0001-7411-3831 <petrenko@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2] *Московский физико-технический институт,*
114701, Россия, Московская область, г. Долгопрудный, Институтский пер., 9
[3] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, г. Москва, Ленинские горы, д. 1
[4] *НИУ Высшая школа экономики,*
101978, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. Рассмотрен широкий круг вопросов теории и практики разработки крупномасштабного программного обеспечения и гибридных программно-аппаратных систем, в том числе операционных систем. Затрагиваются вопросы управления конфигурацией, моделирования и верификации таких систем, построения онтологических моделей предметных областей, связанных с прикладным и системным программным обеспечением. Такое многостороннее рассмотрение необходимо для обеспечения надежности, безопасности и элволюционного развития в течение многолетнего периода эксплуатации инфраструктурных и критически важных систем. Статья основана на материалах исследований, выполненных в рамках двух проектов РФФИ. Помимо недавних результатов, авторы уделяют внимание истории развития исследований в соответствующих областях в Советском Союзе.

Ключевые слова: моделирование; программирование; повторное использование; вариабельность; безопасность; защита; надежность; качество

Для цитирования: Лаврищева Е.М., Петренко А.К. Моделирование операционных, программных и технических систем в проектах РФФИ. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 265-280. DOI: 10.15514/ISPRAS–2021–33(6)–19

Благодарности: Данная работа выполнена при поддержке гранта РФФИ 19-01-00206.

Modeling of operational, software and technical systems in RFBR projects

[1,2] *E.M. Lavrischeva, ORCID: 0000-0002-1160-1077 <lavr@ispras.ru>*
[1,3,4] *A.K. Petrenko, ORCID: 0000-0001-7411-3831 <petrenko@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2] *Moscow Institute of Physics and Technology (MIPT),*
9, Campus per., Dolgoprudny, Moscow region, 114701, Russia
[3] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia
[4] *National Research University, Higher School of Economics*
20, Myasnitskaya ulitsa, Moscow, 101978, Russia

Abstract. A wide range of issues of theory and practice in the development of large-scale software and hybrid software intensive systems, including operating systems, is considered. Issues of configuration management, modeling and verification of such systems, construction of ontological models of subject areas related to application and system software are touched upon. Such multilateral consideration is necessary to ensure reliability, security and elastic development during the multi-year period of operation of infrastructural and mission-critical systems. The paper is based on the materials of studies carried out within the two RFBR projects. In addition to recent results, the authors pay attention to the history of the development of research in the relevant areas in the Soviet Union.

Keywords: modeling; programming; reuse; variability; security; protection; reliability; quality
For citation: Lavrischeva E.M., Petrenko A.K. Modeling of operational, software and technical systems in RFBR projects. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 265-280 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–19

Acknowledgements. This work was supported by the RFBR grant No. 19-01-00206.

1. Введение

Программирование прошло долгий путь с 50-х годов XX века до наших дней; в результате накоплен опыт создлания разнообразных системных и прикладных программ, в частности: операционных систем, библиотек математических методов решения физико-технических, прикладных и предметных задач для прикладных областей (физика, биология, медицина, химии, генетики и др.). Широкое применение получили функциональные компоненты многоразового использования (КПИ) в разных областях знаний и метод конвейерной сборки технических средств (устройств, приборов) и робототехнических изделий в биотехнике, медицине, химии и др. Каждый ресурс (модуль, объект, компонент) и системы из них проверялись на безопасность, надежность, качество и защиту обработки данных. Международное сообщество сформировало много стандартов по разработке, оцениванию и управлению созданием сложных систем: АИС, АСНИ, АСУ, АСУТП и др. Созданы и действуют робототехнические устройства для проведения физиологических, кардиологических, травматологических, онкологических исследований и проведения хирургических операций. В университетах, институтах и школах страны сформировались программы обучения студентов современным основам программирования и инженерии разработки разных математических задач, технических и операционных систем.

Одним из основных направлений создания программно-технических систем (ПТС) является теория математического моделирования сложных систем из готовых ресурсов, отражающих сформировавшиеся знания в виде интеллектуальных, информационных, операционных и вычислительных средств. В мировом сообществе в рамках информатизации с 1992 созданы специальные информационные технологические инструменты (E-science, Semantic Web, Grid, Etics и др.), обеспечивающие приобретение и извлечению знаний, накопленных в

разных предметных областях знаний для их использования. Одним из средств концептуализации знаний Semantic Web является онтологический аппарат представления знаний в научных, технических, физических, математических, биологических областях.

Начиная с 2001 года, проводились международные конференции VaMoS (International Working Conference on Variability Modelling of Software-Intensive Systems), сформировавшие новые методы моделирования ОС и моделирования задач технических и программных областей: физических экспериментов, моделирования медицинских, экономических, социальных, финансовых задач. Одним из путей решения таких задач является технология производства программных продуктов (ПП), удовлетворяющая определенным экономическим критериям по организации моделирования, разработки и изготовления продуктов высокого качества и производительности с использованием готовых ресурсов, КПИ со стандартными или унифицированными интерфейсами (CALL, RPC, RMI, IDL, WSDL, API, ABI), способствующих проведению формализованной сборки ресурсов в ЯП 4 поколения (C++, Java, Python, Ruby и др.) на различных программных платформах.

2. Периоды развития средств моделирования и программирования на ЭВМ в СССР

2.1 Начальный период моделирования ОС ПО, программных и технических систем

На первых ЭВМ -МЭСМ (1955), СТРЕЛА (1956), М-20 (1958) были реализованы ОС и ПО для проведения работ по трансляции, отладки, тестированию программ на первых языках программирования ЯП Algol-60, Fortran, PL/1, Cobol и др. Программы вводились с внешних устройств ввода, вывода и ОС управляла их обработкой (трансляция и др.) на ПО (трансляция и др.) с сохранением результатов во внешней памяти ЭВМ. Для трансформации информации в код ЭВМ использовались трансляторы (ТА1 –ТА4) с ЯП, отладчики, тестировщики, сборщики и др. Первыми программам были простые численные задачи с данными для вычисления. Результаты запоминались в библиотеках программ. По решению ГКНТ СССР были созданы Республиканские фонды алгоритмов и программ (РФАП), так называемые склады программ, и разработан метод объединения, сборки отдельных программ в разных ЯП в большие системы для их выполнения в операционной среде ЭВМ. Одновременно программировались разные технические, математические, интеллектуальные и информационных задачи для постановки и решения на ЭВМ.

2.2 Глобальные цели для инициации процессов моделирования технических систем

Понятие *модель* первоначально сформировалось в 1613 году при строительстве зданий в Древней Греции в виде образца здания и модели задачи Аль Хорезми и Ньютона для численных расчетов задач на аналитических машинах. Моделирование техники развивалось при строительстве зданий, мостов, лодок, самолетов и др. Модель отражала сущность технического предмета или явления и связи элементов. В 18-19 веке в России созданы технические средства -радио, телефон, самолет, автомобиль… Значительный скачок развития технических средств и прикладной математики произошел после окончания Великой Отечественной войны (ВОВ) 1945 года по следующим направлениям: – передовые систем вооружений, обусловленные ходом ВОВ;–гонка вооружений между СССР и США; – ядерная программа И.В. Курчатова после атомного взрыва в Хирасимо и Нагасаки; – ракетостроение, аэродинамика, баллистика, космос и др.; –электронная техника, радиосвязь и численное решение уравнений Ньютона, Лейбница, Максвелла и др.; –активная разведка природных ресурсов, добыча нефти и газа в нашей стране.

2.3 Моделирование техники и математики

Важным стимулом к развитию теории практики программирования стали задачи создания технологий военного назначения. Сразу после окончания Второй мировой войны Министерство Радиопромышленности СССР создало программу развития радиотехнических и радиофизических приборов с обеспечением высокого качества и безопасности. В рамках предприятий военно-промышленного комплекса (ВПК) были созданы специализированные устройства и средства радиотехнического, радиолокационного, авиационного, космического назначения. Была создана атомная бомба и космический корабль с Ю. Гагариным борту (1961). Весь мир был поражен уровню достижений советской науки. Были созданы специализированные ЭВМ (ПРА-6.0, МАПА, АРГОН, АОУ6 и др.), а также радиолокационная техника для наведения и слежения за движением самолетов, подводных лодок, ракет, кораблей и др. Созданы программные комплексы ПРОТВА, ЯУЗА, РУЗА, ПРОМЕТЕЙ для решения задач противовоздушной обороны и других задач оборонной тематики (подробнее см. [1] и другие книги В.В. Липаева).

2.4 Моделирование математических задач энергетики, баллистики и геофизики

Ядерный взрыв в Хирасимо и Нагасаки в 16 июля 1945 года послужил поводом для создания в нашей стране Атомного реактора под руководством И.В. Курчатова и проведения многочисленных расчетов, связанных с распадом ядерного урана и плутония для ядерной бомбы. Была создана теория цепной реакции распада урана и формирования математических методов в ядерной энергетике, что привело к созданию в стране новых энергетических и технических устройств (РДС-3-7), атомной бомбы в 1949 году, первая АТС и атомный ледокол им. Ленина (1950). После развала идет создание Термоядерного реактора у нас и Европейского проекта коллайдера и др. Моделирование методов в энергетике и баллистики активно проводились на первой МЭСМ, созданной член-корр. С.А. Лебедевым (1948-1951) коллективом лаборатории Института электротехники АН УССР. На МЭСМ под руководством академика М.В. Келдыша и с участием академика А.А. Дородницына решались математические задачи внешней баллистики, электротехнические задачи, задачи теории упругости; обеспечения устойчивости энергосистем, задачи расчета тепловых напряжений строительных конструкций, задачи выбор оптимальных параметров шахтных канатов, геодезические задачи, оценка объемов земляных работ автодорог (и БАМ) и многие другие ([2-4].

2.5 Моделирование физических задач и механики сплошных сред

Проведение вычислительных экспериментов с помощью математической модели физического эксперимента, включающей характерные свойства физического явления, алгоритма свойств и характеристик этого явления, задаваемых в виде простых формул и операций программы решения задач (математической физики, нелинейной механики и сплошных сред и др.) на ЭВМ. Одними из первых важнейших вычислительных задач была задачи о ядерном взрыве и об обтекании тел сверхзвуковым потоком газа. Исследование физических объектов проводилось путем анализа и выбора схем эксперимента для определения элементов физических устройства. Затем составляется разностная схема и проверка ее устойчивости. При математическом моделировании этих задач создавалась приближенная модель. Разработка рациональных численных моделей проводилась с помощью интеллектуальных гибридных э*кспертных систем для* прогнозирования поведения объектов и *распознавания* образов объектов с учетом временных интервалов. Проблема *интеллектуализации* экспертных систем была перенесена на решение задач поиска динамических инвариантов для соответствующих классов эволюционных задач. При моделировании численных задач в механике сплошных сред реализованы методы

дискретных вихрей, Монте-Карло, метод неопределенных коэффициентов стационарных методов и расщепления физических процессов с сильно изменяющимися границами области интегрирования [5, 6].

2.6 Первые проекты системного программирования в СССР

Первыми примерами системных программ были прообразы операционных систем, компиляторов, редакторов внешних связей и загрузчиков, что несколько позже стало называться системами программирования. Среди наиболее значимых результатов следует отметить:

- Диспетчеры и операционные системы для МЭСМ, М-20, Стрела, Весна (М.Р. Шура-Бура, В.С. Штаркман), БЭСМ Д-68 и НД-70 (Л.Н. Королев, А.Н. Томилин, В.П. Иванников и др.),
- ОС ИПМ для М-220 1967 г. ИПМ (И.Б. Задыхайло, С.С. Камынин и др.),
- Монитор -Дубна (Н.Н. Говорун и В.П. Шириков).
- Программирующая программа для языка операторных схем Ляпунова А.А. сделана в 1953 г. в МГУ (Э.З. Любимский, А.П. Ершов) и трансляторы с языка Алгол-60 (ТА) для работы с программами на этом ЯП:
- ТА1–С.С. Лавров (ЛГУ, 1962); ТА2–М.Р. Шура-Бура и Э.З. Любимский (ИПМ, 1963);
- ТА3 -(Альфа-система) версия языка Алгол-60 (СО АН СССР, 1964);
- ТА4 – Е.Л. Ющенко, Е.М. Лаврищева для УВК Днепр-1, Днепр-2 (ИК АН УССР).
- Транслятор с языка АЛМО (Любимский Э.З., 1965).
- Первая Библиотека стандартных математических подпрограмм ИС-2 для М20 в МГУ (Е.А. Жоголев, Г.С. Росляков, Н.П. Трифонов, М.Р. Шура-Бура. Система стандартных подпрограмм) и др.
- Алгоритмизация задач прикладной математики для первых ЭВМ с использованием теории граф-схемного языка А.А. Ляпунова (1955). Описание математических элементов задач с помощью математических и логических операций и операторов перехода ориентированного графа с одной входной и одной выходной вершиной.

2.7 Теория графового моделирования задач техники и математики

Впервые теория графов была открыта Эйлером в 1736 г. и описана в статье про строительство Кенигсбергских мостов. В течение почти ста лет эта теория мало использовалась. Интерес к теории графов возродился в середине XIX в связи с развитием естественных наук (электрических сетей, моделей кристаллов, структур молекул и др.) и математической логики. Решение многих математических головоломок, шахматных задач формулировались в терминах графовой теории. Последние 50 лет ознаменовались использованием теории графов в атомной энергетике, нейронных. нанобиологических, генетических, геофизических, космических задачах и др., например, физический эксперимент Европейского проекта «Коллайдер», ядерный реактор, компьютеры, информатика, телекоммуникация, экономика и др.

Теория графов в программировании: А.С. Ляпунов (МГУ), А.П. Ершов, Касьянов В.И, Иткин В.Э., Евстигнеев А.А. и др. (НГУ) и др. Эта теория активно развивается в компьютерных науках и при моделировании систем. Математический аппарат инцидентности и смежности использовался для доказательства графовых, нейрографовых структур и др. (Апонович, НГУ). Графовые структуры используются при проведении физических экспериментов, разработке информационных и интеллектуальных систем. В ИСП РАН графовая теория применялась при моделировании вариантов OS Linux (2018-2021) с обеспечением надежности и безопасности. Подробнее с этим направлением можно ознакомиться по работам [7-14].

2.8 Развития информационных и интеллектуальных систем

Начиная с 2000 годов, в мировой сообществе проводились конференции VAMOS 2002-2021. Многие компании по производству программных продуктов проводили практические работы по созданию ПП и возможности изменения отдельных программ в действующих системах и генерацию вариантов продуктов. Software Engineering Institute of USA разработал концепцию продуктовую линию производства продуктов ПС и СПС – Product Line/ Product Family. Продуктовая линия позволяет собирать разные варианты систем из готовых программных элементов (программ, компонентов, assets и др.) под требования потребителей. Создавались варианты ПС и СПС небольших размеров с требуемым количеством функций. В результате сформировался метод сборки вариабельных продуктов и систем разного назначения. Основу метода составляла модель внешних характеристик MF (Feature Model) функций с заданием интерфейсов связи с другими функциями системы. Появился стандарт конфигурационной сборки IEEE 828: Configuration-2009 функциональных элементов в структуру системы. Стандарт позволял манипулировать характеристиками системы для создания разных вариантов выпускаемых при производстве систем разного назначения.

2.9 Вариабельность систем

Модель вариабельности MF включает оценочною модель для сбора и накопления ситуаций, возникающих при разработке и сопровождении семейств СПС, документирования этих ситуаций в терминах значений уровней изменяемости и степени соответствия системы потребностям предметной области. Для управления вариабельностью используется фреймворк *COVAMOF* (ConIPFVariability Modeling Framework), который погружается в среду MS VisualStudio моделирует создание ПС, СПС по модели MF и архитектуре из готовых компонентов (рис. 1). Основные публикации: [15-20].

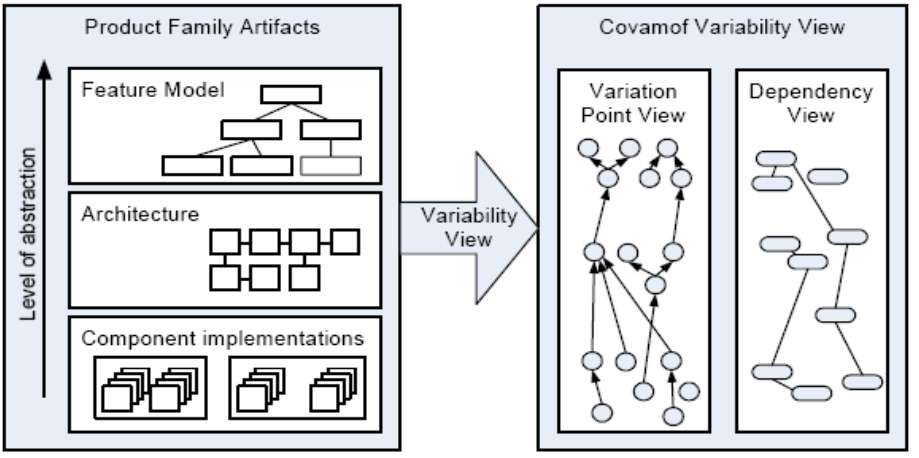


Рис. 1 Обработка на вариабельность моделей и компонентов
Fig. 1 Processing for variability of models and components

2.10 Моделирование онтологии систем в Семантик Веб

В Семантик Веб для поддержки онтологии имеются инструменты: Protege, Eclipse, основанные на фреймовой модели представления знаний и редактирования моделей в UML, XML, SHOE, DAML, OML, DSL, RDF и RDFS. В Protege имеются готовые плагины: Protege OWL Plugin подобный тестам JUnit; средства генерации описаний в языке Java и возможность преобразовывать в формат XML-схемы на платформе Eclipse в XML Schema Infoset Model (XSD). Онтология приложения, ПС, СПС описывается с помощью классов, слотов, фасетов и аксиом. Классы содержат описание базовых понятий предметной области, слоты

определяют свойства понятий, Фасеты описывают свойства слотов (типы и диапазоны значений). Аксиомы определяют ограничения (правила) и отношения. Классы могут быть абстрактными или конкретными. Абстрактные классы являются контейнерами конкретных классов и могут содержать абстрактные атрибуты. Конкретные классы содержат слоты, которыми могут быть значения атрибутов. Фасеты и слоты задаются графовыми XML-схемами. Кардинальный слот определяет возможное количество значений слота, ограничение типа значений (например, целое, буквенное и др.), предельные значения (мин. и макс.) для числовых слотов и т.п. Средствами онтологии описывается структура домена ЖЦ стандарта ISO/IEC 12207 с выходным результатом в виде графа XML-структуры.

2.11 Онтология ЖЦ ISO/IEC 12207-2012

Стандарт ISO/IEC ЖЦ 12207–2012 является основным инструментом планомерного процессного изготовления ПС. Он включает три категории процессов:1) основные процессы; 2) процессы поддержки; 3) организационные процессы. Эти процессы включают способы и форм представления и выполнения результатов (рис.2, 3).

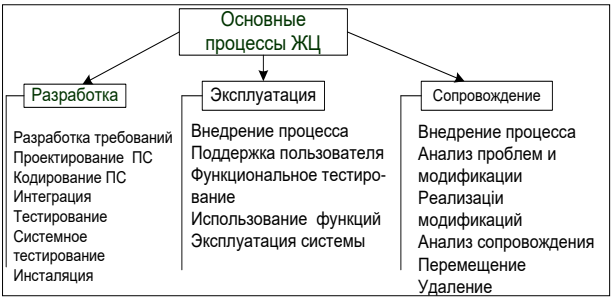


Рис.2. Схема основных процессов ЖЦ ПС для онтологии
Fig.2. Scheme of the main processes of the life cycle of PS for ontology

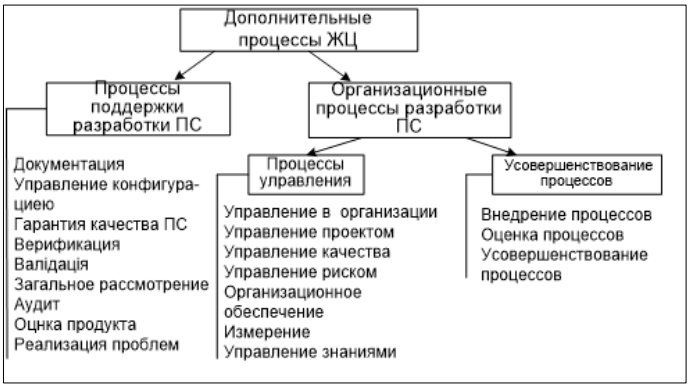


Рис. 3. Схема вспомогательных процессов ЖЦ ПС
Fig 3. Scheme of auxiliary processes of the life cycle of the PS

Стандарт жизненного цикла содержит в себя также вспомогательные процессы, которые регламентируют дополнительные действия при проверке продукта, управлении проектом и качеством. Как правило, в зависимости от целей конкретного проекта главный разработчик и менеджер проекта выбирают процессы, действия и задачи, выстраивая определенную модель жизненного цикла для этого проекта. Данный стандарт (в варианте 2007 года) включает в себя (табл. 1) 17 процессов, 74 подпроцесса и 232 технологические операционные задачи (действия). Их необходимо и достаточно для проектирования систем с помощью процессного

подхода. Некоторые системные фирмы производители программных пакетов реализуют отдельные фрагменты или варианты этого стандарта.

Табл. 1. Процессы, подпроцессы и задачи жизненного цикла
Table 1. Processes, sub-processes and life cycle tasks

Классы	**Процесс**	**Действие**	**Задача**
Основные процессы	5	35	135
Процессы поддержки	8	25	70
Организационные процессы	4	14	27
Всего	17	74	232

Концепция автоматизации стандарта жизненного цикла средствами онтологии является новой. Базовыми понятиями онтологии являются процессы жизненного цикла, их взаимосвязи по передаваемым результатам проектирования программных систем, а также функции процессов для их реализации и выполнения. Если все 17 процессов будут реализованы, то можно из них генерировать некоторые подмножества жизненного цикла, как вариант рабочего жизненного цикла для конкретного применения. Нами рассмотрена онтология процесса тестирования.

Для проектирования онтологии жизненного цикла могут использоваться языки BPMN и BPEL. Жизненный цикл представляется с помощью словарей понятий, концептов и отношений между ними в среде Protégé. Полученное онтологическое описание жизненного цикла трансформируется на язык XML, который при реализации размечает данные домена жизненного цикла и устанавливает связи и обмен данными между процессами.

Домен ЖЦ занимает центральное место в программной инженерии, основным назначением которого является поддержка методов и средств изготовления сложных программных систем. Изучающие эти методы и средства, а также современные стандарты ЖЦ ISO/IEC 12207–2007 и ISO/IEC 11404–2007 GPD (General Purpose Datatypes) легли в основу разработки экспериментального варианта онтологии ЖЦ. Концепция онтологии ЖЦ обсуждалась в КНУ на научных семинарах кафедры теории программирования и технологии, кафедры информационных систем и в МФТИ на кафедре информатика и прикладная математика. Приводимое описание онтологии получено с участием студентов.

2.12 Моделирование онтологии «Вычислительная геометрия» в Семантик Веб

Задачи вычислительной геометрии являются классическими, поэтому эту область можно использовать как хороший пример демонстрации построения онтологий. К основным понятиям вычислительной геометрии относятся:

- выпуклая оболочка с набором точек, необходимых для нахождения наименьшего выпуклого многоугольника, содержащего все точки;
- пересечение отрезков в наборе отрезков;
- триангуляция Делона;
- диаграмма Вороного, содержащая набор точек, разделяющих пространство на сектора, каждая точка которых ближе к набору других;
- задача ближайшей пары точек из набора точек для нахождения кратчайшего пути;
- евклидов кратчайший путь, который соединяет две точки в Евклидовом пространстве (с полигональными препятствиями) кратчайшим образом;
- триангуляция многоугольника и разбивка на внутренние треугольники;
- метод Грехема и Джарвиса для обхода точек в простых и выпуклых многоугольниках; метод быстрой сортировки на множестве точек и линий пересечения фигур;
- метод М.Шеймоса организации данных для выпуклых многоугольников;

– графовые структуры Штейнера для решения задач триангуляции и поиска соседа.

Рассматриваются статические и динамические задачи. К статическим задачам относится:

- *выпуклая оболочка*: имея набор точек необходимо найти наименьший выпуклый многоугольник, который содержит все точки;
- *пересечение отрезков*: найти все пересечения в наборе отрезков;
- *триангуляция Делона*;
- *диаграмма Вороного*: имеется множество точек на плоскости, требуется разделить это множество на области, чтобы каждая такая область образовывала множество точек, более близких к одному из элементов исходного множества, чем к любому другому его элементу;
- *задача ближайшей пары точек*: имея набор точек найти кратчайшее расстояние;
- *евклидов кратчайший путь*: соединить две точки Евклидова пространства (с полигональными препятствиями) кратчайшим образом.
- *триангуляция многоугольника*: имея многоугольник, разбить его на внутренние треугольники.

Задачи геометрического поиска включают пространство поиска:

- *региональный поиск*: обработать набор точек, с целью эффективного поиска набора точек, содержащихся в заданном регионе.
- *локализация точки*: имея разбиение пространства на регионы, создать структуру данных, которая позволит эффективно определить, в каком регионе находится данная точка.
- *поиск ближайшего соседа*: обработать набор точек чтобы иметь возможность эффективно найти точки, близкие к заданной.
- *трассировка лучей*: для заданного набора объектов в пространстве создать структуру данных, которая позволит эффективно определять объекты, пересекающие заданный луч.

Комбинаторная вычислительная геометрия в терминах базовых геометрических объектов: точки, отрезки, многоугольники, многогранники. Численная вычислительная геометрия или машинная геометрия.

2.13 Метод сборки информационных и интеллектуальных ресурсов в ПС и СПС

В начале 70-х годов сформировалось понятие «модуля» как главного элементарного, программного элемента, выполняющего некоторую отдельную функцию в общей архитектуре программной системы. Модуль описывался в ЯП (Algol-60, Fortran, Cobol, Smalltalk) и переводился в код ЭВМ отечественными трансляторами в среде ОС ЭВМ. Модули образуют элементы систем и накапливаются в репозитории ОС ПО. Академик В.М.Глушков высказал идею о том, что программы из модулей будут собираться конвейерным способом, как автомобили из готовых деталей на фабрике Форда. Реализован метод сборки модулей на основе межмодульного интерфейса (1976–1982) в системе АПРОП*. Разработан межмодульный, межъязыковый и технологический интерфейс. Получена грамота за интерфейс на конференции «Интерфейс СЭВ (1987) и он стал базовым понятием в технологии программирования.

Метод сборки и библиотека интерфейса (64 примитивов) реализованы в ОС ЕС/IBM360, внедрен в ВПК (В.В. Липаев, МНИИПА) для создания комплексов РУЗА, ПРОМЕТЕЙ, ЯУЗА и специализированных ЭВМ (ПРА-6.0, МАПА, АРГОН, АОУ6 и др.). Система АПРОП была передана в 52 организации СССР и отмечена Государственной премией Кабинета министров СССР (1985). Основные публикации: [18-20].

2.14 Теория сборочного программирования

Сущность теории решения задачи сборки пар разнородных модулей или объектов состоит в построении взаимно однозначного соответствия между множеством формальных и фактических параметров [18].

$V = \{v^1, v^2, ..., v^\kappa\}$ вызов объекта с множеством формальных параметров;

$F = \{f^1, f^2, ..., f^{\kappa 1}\}$ вызов объекта и их отображение с помощью алгебраических систем [2–4].

Каждому типу данных T_α^t языка l_α ставится в соответствие алгебраическая система вида:

$$G_\alpha^t = <X_\alpha^t , G_\alpha^t >,$$

где X_α^t – множество значений рассматриваемого типа данных (ТД), а G_α^t – множество операций над объектами данного типа. Операциям преобразования ТД соответствует изоморфное отображение алгебраических систем G_α^t в G_β^d.

В классе алгебраических систем

$$\Sigma = \{G\ \alpha^b , G\alpha^c , G\alpha^i , G\alpha^r , G\alpha^a , G\alpha^z\},$$

где тип t – это b – boolean, c – character, i – integer, r – real, a – array, z – record и др. В системе Σ реализованы все виды преобразований для представленных ТД и проведено доказательство изоморфизма алгебраических систем [19] и его отсутствие для множеств неэквивалентных значений X_α^t и X_β^q. Установлено, что мощности алгебраических систем равны

$$|G_\alpha^t| = |G_\beta^q|.$$

В парадигме сборки реализованы универсальные формальные основы преобразования типов входных / выходных данных (простых и сложных) FDT к общим типам данных GPD стандарта ISO/IC 11404 (примитивные, агрегатные, генерированные и неструктурированные) [1–10]. Ниже рассматриваются общие вопросы преобразования FDT, GPD и GPD<=>FDT, как очень важные для сборки программ для среды Интернет.

Теории объектно-компонентного моделирования посвящены следующие публикации: [18-20].

2.15 Генерация ТД стандарта GPD

Разработана схема генерации ТД GPD в структуры фундаментальных ТД ЯП. Предлагается разработать библиотеку функций генерации ТД стандарта GPD, элементы которой для всех новых ТД этого стандарта выполняют следующие виды операций:

- преобразование ТД, содержащихся в ЯП ($ЯП_1$, …, $ЯП_n$) и входящих в состав ФДТ ТД;
- функции трансформации сложных ТД стандарта GPD, включенных в стандартную библиотеку CTS VS.Net и используемых трансляторами с ЯП этой системы для преобразования сложных ТД к более простым;
- операции взаимодействия компонентов повторного использования, записанных в разных ЯП, интерфейс которых задается в языке IDL.

Все ТД стандарта GPD представлены в виде следующих классов алгебраических систем:

$$\sum 1 = \{G\alpha^b , G\alpha^c , G\alpha^u , G\alpha^r\},$$
$$\sum 2 = \{G\alpha^a , G\alpha^z , G\alpha^u , G\alpha^e\},$$
$$\sum 3 = \{G^s , G^{ns}\},$$

где $G\alpha^t = <X\alpha^t, \Omega\alpha^t>$, t – ТД языков L, $X\alpha^t$ – множество значений ТД; $\Omega\alpha^t$ – множество операций над ТД; $\sum 1$ - простые ТД: $t = b$ (bool), c (char), i (int), r (real)), $\sum 2$ – сложные (структурные) типы данных: t = a (array), z (record), u (union), e (enum).), как комбинации простых ТД; $\sum 3$ - сложные, неструктурированные ТД (портфель, контейнер, протокол и т.п.) и сгенерированные из них более простые данные.

В каждом классе этих систем преобразование $t \rightarrow Tq$ для пары языков lt и lq основано на таких свойствах отображений:

1) системы $G\alpha^t$ и $G\beta^q$ – изоморфны, если их q, t определены на том же множестве ТД;
2) между значениями $X\alpha^t$ и $X\beta^q$ типов данных t, q существует изоморфизм, если множество операций $\Omega\alpha^t$ и $\Omega\beta^q$ разные;
3) если множество $\Omega = \Omega\alpha^t \cap \Omega\beta^q$ не пустое, то имеет место изоморфизм двух систем;
4) $G\alpha^{t'} = < X\alpha^{t'}, \Omega >$ и $G\beta^{q'} = < X\beta^q, \Omega >$;
5) если типы данных отличаются, например, t - строка, а тип q – вещественное, то между множествами $X\alpha^t$ и $X\beta^q$ не существует изоморфного соответствия.

Отображения сохраняют линейный порядок элементов к виду линейной упорядоченности элементов алгебраических систем из этих классов.

2.16 Теория технологии программирования (ТП)

Теорию ТП как математическую дисциплину сформулировали А.П. Ершов и зарубежные ученые (E. Dijkstra, A. Hoar, Z. Manna, D. Gries, D. Bjorner, K. Parnas др.). Так в докладе на Всесоюзной конференции «Технология программирования» (1986) А.П. Ершов определил:

Теория ТП –это методы синтеза, сборки и конкретизации.

- *Синтезирующее программирование* базируется на методе доказательного рассуждения о правильности программы.
- *Сборочное программирование* осуществляет построение программы из уже существующих (проверенных на правильность) готовых фрагментов программ (reuses) и сборку их в сложную структуру.
- *Конкретизирующее программирование* обеспечивает построение системы по универсальной модели для некоторой предметной области.

Программные элементы требуется верифицировать, тестировать для поиска ошибок и их исправления, а затем после интеграционного тестирования оценивать на качество (см. [21, 22].

2.17 Методы и средства обеспечения надежности и качества технических, системных и программных средств

В рамках проекта РФФИ 19-01-00206 решались задачи моделирования качественных прикладных, операционных систем и сборки экспериментального варианта ядра OS Linux для применения в комплексах, технике и приборах для областей знаний. Для решения задач сборки варианта OS Linux был проведен анализ свойств используемых приборов с позиций обеспечения надежного функционирования элементов ОС с помощью операций сборки (build, assembly, config, make, waver) функций OS Linuxв новое экспериментальное ядро для областей знаний. Операции сборки реализованы в современных операционных средах: makeв BSD и GNU; build, assemblyв JavaEE, Grid, Etics, IBMSphere, VS.MS, Intel, Unix, Linux и др. Результаты исследований и реализации задач моделирования прикладных и операционных систем с обеспечением качества представлены в работах:

Технология сборки экспериментального варианта ядра OS Linux с обеспечением качества включает следующие процессы.

1) Препроцессирование – предварительная обработка функции ядра ОС, компиляция для получения кода версии с поиском ошибок.
2) Компиляция формально описанных функций для получения выходного кода с проверкой ошибок в синтаксисе их описания.
3) Извлечение функциональных элементов ядра и представления их в классы эквивалентности с проверкой, тестированием MC/DC.

4) Связывание (сборка) выделенных элементов ядра с помощью операторов make, config и интерфейса связи элементов и обмена данных между ними из библиотек Интернет.
5) Конфигурация структуры с помощью операции Kconfig для получения файла конфигурации системы.
6) Тестирование используемых элементов и собранного из них варианта ОС с поиском ошибок с помощью LDV и CPAchecker и оценка правильности компонентов ОС.
7) Оценивание элементов сконфигурированного варианта ядра ОС на надежность, безопасность и защиту информации.

Стандартные средства обеспечения безопасности ОС:

– криптография (CriptographicApplication Program Interface – CAPI), включающая контроль ключей, информации и доступа к данным;
– защита данных в системах общего назначения, в состав которых входит система безопасности, ориентированная на противодействие угрозам безопасности и защиты информации в компьютерной сети (КС). Система безопасности обеспечивает:
– санкционированный доступ к объектам КС;
– успешное отражения угроз (внешних или внутренних) в сети;
– контроль прохождения запросов в сети и оценки эффективности безопасности и др.

Модель системы безопасности имеет следующие характеристики:

– авторизация (АВТ), заключающаяся в типизации пользователей или их групп для получения разрешения на совместное использование отдельных ресурсов ПТС;
– аутентификация (АУТ), устанавливающая подлинность личности, посылающей сообщение в сеть для доступа к ресурсу;
– парольная защита (ПЗ) и контроль доступа к информации, программам и устройствам.

Для оценки надежности экспериментальной версии ядра OS Linux использовался метод вероятностной надежности Маркова и методы оценки полноты верификации. Результаты этого направления были опубликованы в следующих работах: [23-26].

В рамках проекта РФФИ №19-01-00206 проведены исследования гарантоспособности с учетом возникающих событий типа аварий и контратак в экспериментальном варианте OS Linux, а также в созданных технических средствах для авиации, промышленности, морского флота и др.

Таким образом, в рамках работ по проекту РФФИ №19-01-00206 сконфигурирован экспериментальный вариант OS Linux для применения в предметных областях знаний (медицина, физика, математика и др.). Созданный вариант ОС проверен на правильность методами верификации, валидании, тестирования и оценки показателей надежности, качества и безопасности. Данный вариант ядра ОС может использоваться для управления процессами решения функциональных задач предметных областей знаний (медицина, физика, биологии и др.).

3. Заключение

Моделирование и программирование технических и математических задач прошли долгий путь развития и сыграли важную роль в истории моделирования и программирования технических, программных, операционных и системных комплексов различного назначения на ЭВМ (1945-2000) в рамках работ гражданской и оборонной тематики.

Созданные и используемые технологии не были стандартизированы у нас в стране, и после 1992 года стали использоваться в основном зарубежные технологии. Это UML (Unified Modeling Language), OWL (Ontology Web Language), WSDL, XML, XSL, RDF, Семантик Веб, WWW3C.

С 2004 года, сформировалось модель (Model Variability) на линиях производства программ Product Line, в Grid-системах и применяется на фабриках программ AppFab. Особое развитие

получил научный сервис в среде Семантик Веб Интернета (www.semantic_web.org), позволяющий производить композитную сборку сервисов (SOA, SCA). Разработаны современные технологии системного программирования, интеллектуальные средства (Data Mining) для анализа данных в Legacy систем, извлечения знаний (Mining knowledge) и данных в разных системах Big Data, Cloud Computing Сети Интернет.

В отечественной практике разработана теория ОКМ моделирования систем и средств преобразования ТД стандарта ISO/IEC11404 GPD и неструктурных ТД Big Data. Отработана конфигурационная сборка информационных, программных и операционных ресурсов в систему и представлен пример сборки экспериментального ядра OS Linux.

Сборочный процесс производства ПП основывается на контроле деятельности исполнителей и оценке показателей качества как отдельных КПИ, так и вариантов продукта (например, экспериментального варианта OS Linux). В мировой практике для этих целей используются следующие технологии программирования и моделирования:

- конвейерная сборка с планированием трудозатрат, учетом, контролем результатов и др.) в автомобильной, авиационной промышленности и др.;
- анализ смысла решаемых задач в предметных, п сложных прикладных областях и формализованное их описание в виде КПИ для многократного использования при сборке сложных ПТС;
- развитие средств моделирования с обеспечением контроля изменяемости (вариабельности), безопасности, защиты и качества;
- технологии E-science, Semantic Web, Grid для проведения физических экспериментов и создания информационных и интеллектуальных областей знаний и решения соответствующих задач;
- совершенствование методов моделирования сложных систем (OOP, UML, MDA и др.).
- стандарты конфигурационной сборки (IEEE 828 -1996: Configuration; IEEE 828-1996: Configuration; ISO 14598:2000: планирование и управление электроникой; IEC 61508-2000: функциональная безопасность и др.)

Наиболее быстрый переход к такой организации работ состоял в систематизации и формализации операций сборки для многократного использование готовых информационных ресурсов, в том числе с нестандартными типами данных в случае Big Data в Cloud Computing, в современной среде Интернет.

Проблематика моделирования вариабельных сложных систем начала исследоваться в ИСП РАН в 2014 году под руководством академика В.П. Иванникова. Методы моделирования ПТС с обеспечением качества, защиты и безопасности были представлены в проектах РФФИ №16-01-00352 «Теория и методы разработки вариабельных программных и операционных систем» (от 24.09.2016) и РФФИ №19-01-00206 «Модели, методы и средства обеспечения надежности программных и технических систем» от 21.02.2020.

Проект РФФИ №16-01-00352-2016 успешно завершен в 2018. В нем принимали участие сотрудники ИСП РАН - Петренко А.К., Карпов Л.Е, Пакулин Н.И., Кулямин В.В., Митулин В.С., Рыжов А.Г.

В проекте РФФИ №19-01-00206 зафиксированы наработки в 2019 - 2021 годов. В проекте принимали участие сотрудники ИСП РАН – Петренко А.К., Митулин В.С., Рыжов А.Г., Зеленов С.В. и др. Проведено исследование и обработка отдельных методов надежности применительно к специальным устройствам и программам (2019-2021). Исследован набор международных методов по проблематике безопасности и защите данных программ и устройств от аварий, узроз и контратак. Эти методы и средства обеспечения безопасности функционирования систем прошли апробацию на отдельных технических, программных и операционных средствах. Методы надежности, качества с обеспечением безопасности и защиты данных описаны и внедрены на задаче создания экспериментального ядра OS Linux.

В исследованиях и в разработке данных проектов принимали участие более 10 ведущих специалистов ИСП РАН отдела «Технологий программирования», а также студенты МФТИ, которые на практике отрабатывали отдельные аспекты теории моделирования сложных систем в 5 магистерских диссертациях. Научные аспекты моделирования систем и обеспечения надежности, безопасности и защиты данных отображены в 25 докладах и статьях на Всероссийских и международных конференциях и в Трудах ИСП РАН.

Результаты исследований проектов РФФИ:

- верификация и тестирование готовых ресурсов (КПИ), используемых при моделировании программных, операционных и технических средств;
- сборка (конфигурация) готовых ресурсов в сложные системы с доказательством их правильности и представления конфигурационной структурой прикладных и OS Linux с помощью операций config, assembly, make, cmake, building;
- отработан метод преобразования разнородных типов данных и систем общих типов данных стандарта ISO/IEC 11404 GPD -1996; GPD-2012 для неструктурированных типов данных Big Data и Cloud Computing средствами SOA, SCA и системных средств Интернет;
- исследованы логические спецификации системных и сервисно-компонентных моделей SOA, SCA, SCM WWW3C с использованием новых функций систем;
- реализован метод сборки информационных и операционных ресурсов с использованием новых операций конфигурирования: config, assembly в Grid; build, waver в Etics: make в BSD, cmake в GNU и JavaEE. С помощью операций make создан экспериментальный вариант ядра OS Linux и Веб-сайта с использованием сервисных ресурсов SOA, SCA, SOAP WWW3C Интернет;
- отработаны методы обеспечения безопасности, защиты данных, надежности и качества ПТС в клиент-серверной среде Интернет на программных структурах систем для применения на сайте ИСП РАН;
- отработан экспериментальный вариант ядра OS Linux, верифицирован, тестирован и проверен на безопасность, защиту данных, борьбу с атаками и отказами и оценкой качественных показателей – функциональность и отказоустойчивость.

Заключительные работы опубликованы в работах [27-29].

Список литературы / References

[1] Липаев В.В. Фрагменты истории развития отечественного программирования для специализированных ЭВМ в 50–80-е годы. Синтег, 2003. 126 стр. / Lipaev V.V. Fragments of the history of the development of domestic programming for specialized computers in the 50–80s. Sinteg, 2003. 126 p. (in Russian).

[2] Кикоин И.К. Игорь Васильевич Курчатов. Атомная энергия, том 14, вып. 1, 1963 г., стр. 5-9 / Kikoin I.K. Igor Vasilievich Kurchatov. Atomic Energy, Volume 14, no. 1, 1963, pp. 5-9 (in Russian).

[3] Марчук Г.И. Численные методы расчета атомных реакторов. Атомиздат, 1958 г., 381 стр. / Marchuk G.I. Numerical methods for calculating nuclear reactors. Atomizdat, 1958, 381 p. (in Russian).

[4] Келдыш М. В. Избранные труды. Общие вопросы развития науки. Наука, 1985 г., 704 стр. / Keldysh M. V. Selected Works. General issues of the development of science. Science, 1985, 704 p.

[5] Тихонов А.Н., Самарский А.А. Об однородных разностных схемах. Журнал вычислительной математики и математической физики, том. 1, вып. 1, 1961 г., стр. 5-63 / Tikhonov A.N., Samarskii A.A. Homogeneous difference schemes. Computational Mathematics and Mathematical Physics, vol. 1, issue 1, 1962, pp. 5-67.

[6] Белоцерковский О.М. Численное моделирование в механике сплошных сред. Наука, 1994 г., 441 стр. / Belotserkovskii O. M. Numerical modeling in continuum mechanics, Nauka, 1994, 441 p (in Russian).

[7] Апонович З.В. Визуализация больших графов и матрица смежности. Труды XX Всероссийской научной конференции «Научный сервис в сети Интернет», 2018 г., стр. 28-41 / Aponovich Z.V.

Visualization of large graphs and adjacency matrix. In Proc. of the XX All-Russian Scientific Conference on Scientific Service on the Internet, 2018, pp. 28-41 (in Russian).

[8] Лаврищева Е.М., Рыжов А.Г. Подход к моделированию систем и сайтов из готовых ресурсов. Труды XX Всероссийской конференции «Научный сервис в сети Интернет», 2018 г., стр. 321-345 / Lavrischeva E.M., Ryzhov A.G. Approach to the modeling of systems and sites from ready resources. In Proc. of the XX All-Russian Conference on Scientific Services on the Internet, 2018, pp. 321-345 (in Russian),

[9] Лаврищева Е.М., Петренко А.К. Технология сборки интеллектуальных и информационных ресурсов Интернет. Труды XXI Всероссийской конференции «Научный сервис в сети Интернет», 2019 г., стр. 469-488 / Lavrischeva K.M., Petrenko A.K. Technology of Assembly of intellectual and information resources of the Internet, In Proc. of the XXI All-Russian Conference on Scientific Services on the Internet, 2019, pp. 469-488 (in Russian)

[10] Лаврищева Е.М. Теория графового моделирования сложных систем из модульных элементов для прикладных областей. Austria-science, no. 28, часть 1, 2019 г., стр. 12-30 / Lavrischeva E.M. The theory of graph modeling of complex systems constructed from modular elements for applied areas. Austria-science, no. 28, part 1, 2019, pp. 12-30 (in Russian).

[11] Lavrischeva E.M., Petrenko A.K., Pozin B.A. Technology of Assembly of Intellectual and Information Resources Internet. In Proc. of the XXI All-Russian Conference on Scientific Services on the Internet, CEUR-WS, vol. 2543, 2019, pp. 247-271.

[12] Lavrischeva E. The theory graph modeling systems from quality modules of the application areas. In Proc. of th6th International Conference on Actual Problems of System and Software Engineering, CEUR-WS, vol. 2514, 2019, pp. 235-247.

[13] Лаврищева Е.М., Петров И.Б. Моделирование технических средств и задач прикладной математики на ЭВМ. Труды ИСП РАН, том 32, вып. 6, 2020 г., стр. 167-182 / Lavrishcheva E.M., Petrov I.B. Modeling of technical means and problems of applied mathematics on computers. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 6, 2020, pp. 167-182 (in Russian). DOI: 10.15514/ISPRAS–2020–32(6)–13.

[14] Лаврищева Е.М., Зеленов С.В. Модельный подход к обеспечению безопасности и надежности Web-сервисов. Труды ИСП РАН, том 32, вып. 5, 2020 г., стр. 153-166 / Lavrischeva E.M., Zelenov S.V. Model-Based Approach to Ensuring Reliability and Security of Web-services. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 5, 2020, pp. 153-166 (in Russian). DOI: 10.15514/ISPRAS–2020–32(5)–12.

[15] Глушков В.М., Лаврищева Е.М. и др. Система автоматизации производства программ (АПРОП). Киев, ИК АН УССР, 1976 г., 134 стр. / Glushkov V.M., Lavrishcheva E.M. et al. Program production automation system (APROP). Kiev, IK AN UkrSSR, 1976, 134 p.

[16] Е.М. Лаврищева, В.Н. Грищенко. Связь разноязыковых модулей в ОС ЕС. М., Финансы и статистика, 1982 г., 137 стр. / E.M. Lavrischeva, V.N. Grishchenko. Linking multilingual modules in the EU OS. M., Finance and Statistics, 1982, 137 p. (in Russian).

[17] Липаев В.В., Позин Б.А., Штрих А.А. Технология сборочного программирования. М., Радио и связь, 1992 г., 287 стр. / Lipaev V.V., Pozin B.A., Shtrikh A.A. Assembly programming technology. M., Radio and communication, 1992, 287 p. (in Russian).

[18] Лаврищева Е.М. Теория объектно-компонентного моделирования программных систем. Препринт ИСП РАН, no. 29, 2016 г., 52 стр. / Lavrischeva E.M. The theory of object-component modeling of software systems. ISP RAS Preprint, no. 29, 2016, 52 p. (in Russian).

[19] Lavrischeva E. Assembling Paradigms of Programming in Software Engeneering. Journal of Software Engineering and Applications, vol. 9, issue 6, 2016, pp. 296-317.

[20] E.M. Lavrischeva. Assembling Paradigms of Programming in Software Engeneering. Journal of Software Engineering and Applications, 9, 296-317.

[21] Ершов А.П. Научные основы доказательного программирования. Вестник АН СССР, no. 10, 1984 г., стр. 9-19 / Ershov A.P. Scientific foundations of evidence-based programming. Bulletin of the Academy of Sciences of the USSR, no. 10, 1984, pp. 9-19 (in Russian).

[22] Ершов А.П. Отношения методологии и технологии программирования. Информационные материалы и тезисы пленарных докладов II Всесоюзной конференции "Технология программирования", 1986, стр. 10-13/ Ershov A.P. Relations between methodology and technology of programming. Information materials and abstracts of plenary reports of the II All-Union Conference on Programming Technology, 1986, pp. 10-13 (in Russian).

[23] Лаврищева Е.М., Пакулин Н.В. Методы и средства обеспечения надежности технических и программных средств. Труды 5-й Научно-практической конференции OS DAY, 2018 г., URL:

http://0x1.tv/20180517F / Lavrishcheva E.M., Pakulin N.V. Methods and means of ensuring the reliability of hardware and software. In Proc. of the 5th Scientific and Practical Conference OS DAY, 2018, URL: http://0x1.tv/20180517F.

[24] Лаврищева Е.М., Пакулин Н.В. и др. Анализ методов оценки надежности оборудования и систем. Практика применения методов. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 99-120 / Lavrischeva E.M., Pakulin N.V. et al. Analysis of methods for assessing the reliability of equipment and systems. Practice of methods. Trudy ISP RAN/Proc. ISP RAS, том 30, вып. 3, 2018 г., стр. 99-120 (in Russian). DOI: 10.15514/ISPRAS-2018-30(3)-8.

[25] Лаврищева Е.М., Зеленов С.В. Модельный подход к обеспечению безопасности и надежности Web-сервисов. Труды ИСП РАН, том 32, вып. 5, 2020 г., стр. 153-166 / Lavrischeva E.M., Zelenov S.V. Model-Based Approach to Ensuring Reliability and Security of Web-services. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 5, 2020, pp. 153-166 (in Russian). DOI: 10.15514/ISPRAS–2020–32(5)–12.

[26] Лаврищева Е.М., Петренко А.К. и др. Технология сборочного создания экспериментального варианта ядра OS Linux с обеспечением качества для применения в прикладных и технических системах. Системный администратор, вып. 11, 2021 г., стр. 80-89 / Lavrischeva E.M., Petrenko A.K. et al. Technology of assembly creation of an experimental version of the OS Linux kernel with quality assurance for use in applied and technical systems. System Administrator, vol. 11, 2021, pp. 80-89 (in Russian).

[27] Лаврищева Е.М., Петров И.Б. Технология моделирования физических и математических задач предметных областей знаний. Евроазиатское научное объединение, no. 1-1, 2021 г., стр. 35-43 / Lavrishcheva E.M., Petrov I.B. Technology for modeling physical and mathematical problems of applied fields of knowledge. Eurasian Scientific Association, no. 1-1, 2021, pp. 35-43 (in Russian).

[28] Lavrischeva E.M., Petrenko A.K., Kozin S.V. Technology of assembly creation of an experimental version OS Linux kernels with quality assurance for applied and subject areas of knowledge. In Scientific Articles Collection of the 81st International Scientific Conference of Eurasian Scientific Association, 2021, pp. 94-102.

[29] Лаврищева Е.М., Петров И.Б., Петренко А.К. Парадигмы моделирования и программирования задач предметных областей знаний: монография. Директ-Медиа, 2021 г., 496 стр. / Lavrischeva E.M., Petrov I.B., Petrenko A.K. Paradigms of modeling and programming tasks of subject areas of knowledge: monograph. Direct-Media, 2021, 496 p. (in Russian)

Информация об авторах / Information about authors

Екатерина Михайловна ЛАВРИЩЕВА – доктор физико-математических наук, профессор, главный научный сотрудник ИСП РАН, профессор МФТИ, лауреат премии Совета Министров СССР (1985). Область интересов: надежность и качество, моделирование сложных систем, Web-системы, сборочное программирование.

Ekaterina Mikhailovna LAVRISCHEVA – Doctor of Physical and Mathematical Sciences, Professor, Principal Researcher at ISP RAS, Professor at MIPT, Laureate of the USSR Cabinet of Ministers Prize (1985). Areas of interest: reliability and quality, modeling of complex systems, Web-systems, assembly programming.

Александр Константинович ПЕТРЕНКО – доктор физико-математических наук, профессор, заведующий отдела ИСП РАН, профессор МГУ и ВШЭ. Область интересов: формальные методы программной инженерии, тестирование программного и аппаратного обеспечения, формальная спецификация требований.

Alexander Konstantinovich PETRENKO – Doctor of Physical and Mathematical Sciences, Professor, Head of the Department of ISP RAS, Professor of MSU and HSE. Areas of interest: formal methods of software engineering, testing of software and hardware, formal specification of requirements.