UDC 004.43

Method of paradigmatic analysis of programming languages

Gorodnyaya L.V. (Institute of Informatics Systems SB RAS, Novosibirsk State University)

The purpose of the article is to describe the method of comparison of programming languages, convenient for assessing the expressive power of languages and the complexity of the programming systems. The method is adapted to substantiate practical, objective criteria of program decomposition, which can be considered as an approach to solving the problem of factorization of very complicated definitions of programming languages and their support systems. In addition, the article presents the results of the analysis of the most well-known programming paradigms and outlines an approach to navigation in the modern expanding space of programming languages, based on the classification of paradigms on the peculiarities of problem statements and semantic characteristics of programming languages and systems with an emphasis on the criteria for the quality of programs and priorities in decision-making in their implementation. The concept of "programming paradigm" is manifested as the way of thinking in the programming process.

The author thanks the organizers and participants of the conferences "Scientific Service in the Internet Environment" (http://agora.guru.ru/display.php?conf=abrau2020&page=subjects&PHPSESSID=qbn3kbhgnk8b6a9g21qi1nkkq2), discussions with which made it possible to understand the main provisions of this article.

This work was partially supported by the Russian Foundation for Basic Research, project No. 18-07-01048-a.

Keywords: *paradigm models, autonomously developed components, teaching system programming, concise definitions.*

1. Introduction

Descriptions of modern programming languages (PL) usually contain a list of 5-10 predecessors and a number of programming paradigms (PP) supported by the language [1,2]. In this article the method of representation of paradigms features of PL definition at the level of semantic systems is considered [3]. Using the method of paradigms analysis it is possible to build a space of constructions supported in the definitions of programming languages and systems (PLS). [4] This space can be the source structure in the selection of criteria of decomposition programs based on the development of statements of problems in the

programming process of their solutions, a variety of types of semantic systems of PL and their extensions in the implementation of programming systems (PS) [5]. The technique is shown on the material of four classical programming paradigms without an excursion into the wider space of paradigms, especially new ones, which have not yet received support in well-known programming languages and recognition in the form of examples of debugged programs. The analysis of DSL—languages, which it makes sense to consider as a new meta-level in the field of programming linguistics, is left for the future. The concept of "programming paradigm" does not have a strict definition [4], so the question arises about the belonging of new approaches in programming to the set of PP and the ordering of such a set [6].

Programming paradigm is manifested as the way of thinking associated with the compromise between the characteristics of tasks, methods of their solution in the form of programs, quality criteria of programs adopted in PP and decision-making priorities in the programming process. Such feature of PP allows to understand a paradigm choice as process of acceptance, representation and debugging of decisions at statement of different tasks therefore it is natural to carry out systematization of PP on comparison with priorities and variations of schemes of statement of tasks and methods of their decision.

The most clear systematization of PP now allows to allocate the basic and derivative PPs supplemented by combined, auxiliary and system-forming or perspective-strategic. It should be noted that academician Andrei Petrovich Ershov was focused on strategic PPs, including fundamental, educational and technological. The set of basic PPs can be divided into basic, instrumentally expanding and unlimited depending on the content of semantic systems of computing organization, memory management, computation management and construction of complex data. The classification of the software can depend on 1) the class and the degree of study of the problems being solved and 2) the potential of the used hardware. .

2. Results of paradigm analysis

Analysis and comparison of a large number of PL of different levels allow to identify the most significant characteristics for the expression of paradigm specificity of a wide class of PL (Table 1).

Table 1. PL twenty-first century (all multi-paradigm)

Year	PL	Predecessors	Used paradigms
2018	Dart	Java, JavaScript, CoffeeScript, Go	object-oriented web application framework script language imperative, reflective, functional
2012	Rust	Alef, C++, Camlp4, Common Lisp, Erlang, Haskell, Hermes, Limbo, Napier, Napier88, Scheme, Newsqueak, NIL, Sather, Ocaml, Standard ML, Cyclone, Swift, C#, Ruby	parallel functional imperative structural systemic procedural free software

2005	F#	OCaml, C#, Haskell	functional object-oriented generalized, imperative
2003	Scala	Java, Haskell, Erlang, Lisp, Standard ML, OCaml, Smalltalk, Scheme, Algol68	functional object-oriented imperative
2001	D	C, C++, C#, Python, Ruby, Java, Eiffel	Imperative, object-oriented, functional, procedural, contractual, generalized
2000	C#	C++, Java, Delphi, Modula-3, Smalltalk	object-oriented generalized procedural functional event-driven, reflective

The multiparadigmality of long-lived and new PLs shows the need for more precise detailing of the dependencies between old and new ones. (Table 2.).

Table 2. PL - the founders of the basic programming paradigms

Year	PL	Used paradigms	Sphere of influence
1954 1958	Fortran, Algol-60	**imperative** parallel procedural modular structural procedural **generalized** object oriented	**IPP** — imperative-procedural ALGOL 58, BASIC, C, Chapel, CMS-2, Fortress, PL/I, PACT I, MUMPS, IDL, Ratfor
1958	Lisp	experimental **functional** object oriented procedural **reflective** **metaprogramming**	**FP** — functional CLIPS, Common lisp, CLOS, Clu, Dylan, Forth, Scheme, Erlang, Haskell, Logo, Lua, Perl, POP-2, Python, Ruby, Cmucl, Scala, ML, Swift, Smalltalk, Factor, Clojure, Emacs Lisp, Eulisp, ISLISP, Wolfram Language
1960	APL	**vector** functional structural **modular**	**PC** — parallel calculation A, A+, FP, J, K, LyaPAS, Nial, S, Wolfram language, MATLAB, PPL,
1962	Simula 67	**object oriented**	**OOP (1980)** — object oriented
1968	Forth	imperative **stack oriented**	Factor, RPL, REBOL, PostScript, Factor and other concatenative languages
1968	Algol-68	**parallel** imperative	C, C++, Bourne shell, KornShell, Bash, Steelman, Ada, Python, Seed7, Mary, S3
1972	Prolog	declarative **logical**	**LP** — logical Visual Prolog, Mercury, Oz, Erlang, Strand, KL0, KL1, Datalog
1970	Pascal	imperative **structural**	**SP** — structural programming Ada, Component Pascal, Modula-2, Java, Go, VHD, Oberon, Object Pascal, Oxygene, Seed7, Structured text

3. Semantic systems of basic paradigms

Considering the systematization of the paradigmatic features of the definition of PL at the level of semantic systems [3], it is convenient to classify language concepts by statement of tasks and language tools used to solve them. Even in last times, Nicholas Wirth noted the importance of matching the problem statement and the tools used to solve it, especially if you can catch the likeness of the processed data structures and their processing algorithms, which is now called homoiconicity. Based on this correspondence, it is possible to build a space of constructions supported in the definitions of PSL and compared with the complexity of the formulations of successfully solved problems. The resulting space can be the initial structure when choosing criteria for decomposing programs, taking into account the peculiarities of the development of problem statements in the process of programming their solutions [4], expanding the semantic systems of PL and their refinement when implementing PS [5].

When considering any semantic systems, it is important to do noted the difference in the nature of the performance of the functions of such systems in different complexes. So, for any data set D representing values from the set V of arbitrary nature, function schemes F are realistically distinguishable for calculation methods, memory access tools M, control features of computing C and communication, or reversible complexation and structuring of data S. This leads to an idea of the main categories of semantic systems for differently implemented types of functions. Historically, at the hardware level, such categories of semantic systems have had a cumulative effects in the “DEMCS” order - the representation of numbers, an arithmometer, a calculator with registers, an analog analizer with control system, a computer. Each hardware subsystem can interact with each other. (Table 3).

Table 3. A number of categories of semantic systems of hardware level.

Subsystem	Note
D: data	Data from set D represents values from the set V and the interrupt scale
E: evaluation	Operations on two or one data produce one or two datum
M: memory	The correspondence between addresses from the set N and representations from the set D stored datum at these addresses allows different methods for accessing memory elements, including replacing stored datum, with the exception of address 0
C: control	Comparing data with zero allows to control the progress of calculations along with go by labels and interrupt handling, not counting the transition in order
S: communications	The construction of complex data takes into account the capabilities of addressing commands in memory

The classification of programming paradigms can depend on 1) the space and degree of knowledge of the problems being solved and 2) the potential of technical devices that support programming paradigms, which reflects the operational and implementation pragmatics of the PL that support these paradigms. As new problems appear, new PPs are formed, the recognition of which by specialists requires a significant time, usually 10-20 years after the emergence of tools. The operational pragmatics of a programming language

strongly depends on the space of the problems being solved and the practicality of their solutions. The implementation pragmatics of programming systems depends on the range of hardware that can be controlled in PL terms (processor, files and peripherals, networks and servers). In terms of the degree of study, the following spaces of problem statements differ significantly, affecting the choice of methods for solving problems and the complexity of their programming:

- new;
- research;
- practical;
- exact.

New problem statements are characterized by the absence of an accessible precedent for solving the problem, the novelty of the means used, or the inexperience of the performers.

Research problem statements are usually complicated by the requirements of originality and versatility, which can be demonstrated in a computer experiment.

Practical problem setting is aimed at relevance and convenience of multiple use of ready-made solutions.

Exact problem statements include testing the limiting capabilities of the tools used, associated with the degree of organization of the created program and the rank of the implemented solutions. The spread of labor intensity, depending on the degree of novelty or knowledge of the problem statement, is usually about 1 to 8. An underestimation of such a spread usually leads to systematic errors in forecasting the forthcoming labor costs.

4. Differences in programming paradigms

Programming paradigms can be distinguished by the priorities of the categories of semantic systems in the programming process, noting the paradigm differences in the general concepts in each category. Data are addresses and stored values representation in IPP, stored methods and object signatures appear in the OOP, be binding with any data in the FP instead of addresses in memory, and to the identifier in the LP. In IPP and OOP, operations are mostly unary or binary, and in FP and LP there is also arbitrary arity. True datum in LP include the special data “ESC”, which allows to distinguish normal predicate values from failure in calculations, and FP can use any data other than “NIL” as truth. Data structures in the IPP can not be considered as values representation processed by the basic means, and in the FP such structures are processed without special restrictions.

When preparing program an IPP, the most important are the means of working with memory in which data and the results of their processing are placed. Data processing is considered as a change in memory states when performing calculations. If necessary, data structures can be organized.

The focus of FP is the organization of calculations on symbolic representations of the entities of a given subject area. Working with memory in this case may not require binding to physical addresses, but rather confine itself to the representation of an associating function over data pairs of any nature. The control of the

computation process can be considered as a function of program fragments. The construction of complex objects is free from the of elements neighborhood.

In the case of LP, the logic of non-deterministic search for feasible solutions dominates. Variants of possible solutions are being choose. Fragments with a fixed number of parameters are named. As structures, samples are used to control the choice of variants.

For OOP, it all starts with defining a hierarchy of classes of objects placed at fixed addresses in memory. The management of the data processing process uses a comparison of classes and valid methods, labeled with access rights from different parts of the program. Computations occur only upon successful matching and matching of access rights to objects. A detailed analysis of the semantics of OOP was performed in [5] and was accompanied by comparison with other software and partial formalization of the main mechanisms.

Thus, in addition to preferences on the features of the problem statements, one can see differences in the schemes for determining functions for different categories of semantic systems depending on the software. It should be noted that the transition from PL to PS is usually accompanied by an increase in the number of supported PPs, which, when defining the Haskell language, led to the concept of "monad", which allows any PL to achieve practicality, which is usually done with the help of library modules.

Description of derivative PPs can be made relative and, therefore, more concise, expressing the difference with the basic paradigm. We can say that the derivative paradigm is a projection of the basic paradigm on the features of the problem statements of a certain application area. Usually in the projection the most important elements of paradigms are modified. Variations of the models of semantic systems that support derivative paradigms can be used as objective parameters in factorizing the definitions of languages and programming systems and decomposing programs, starting with taking into account the peculiarities of problem statements.

For practice, it is useful to describe the derivatives of PP relative, expressing the difference with the base PP. So, IPP derivatives distinguish different methods of representing data in memory and organizing sequential processes generated by the program, OOP derivatives give various concretizations of the concept of "class of objects", FP derivatives represent variations in the methods of organizing calculations, and LP derivatives may use different approaches to mitigate the dependence of obtaining results on excessive or insufficient determinism.

In addition to the relatively clear classical basic paradigms of programming, there is reason to single out the main expanding system-instrumental paradigms aimed at the preparation and design of programs, operating systems and databases, support for working with files and various device configurations, as well as providing feedback when executing any programs. All expanding paradigms, some of them have not yet received their names, work with much more complex elements that have their own lives, which can be included in many systems and configurations in which their state can be changed. Data representations, in addition to complex data structures, formal definitions and codes, include processes, devices, roles of participants and complexes. The methods of processing elements and their interactions are subject to more stringent requirements of correctness, which entails supporting the improvement of elements in parts, that is,

targeted development as errors are identified or the need for increased efficiency. There is a division of labor according to skill level and responsibility.

No less noticeable is the group of unlimited communication interface paradigms supporting large data processing (bigdata, sematic-web, rdf), remote work in networks, service-oriented programming based on markup and rewriting languages (html, XML, PHP), parallel, vector-oriented for processing arrays (APL) or supporting multiple theoretical insertion mechanisms, including dynamic insertion substitutions (SETL) and high-performance computing on supercomputers (OpenMP, mpC) and mobile devices.

There is a noticeable number of combined PPs that combine the advantages of a pair of PPs for solving different types of subproblems, which also are supported by multi-paradigmal PLs (Lisp 1.5, Planner, Merlin, F #, C #, Scala, etc.). There are rejected PPs that have not received recognition by the programming community, and esoteric PPs, the invention of which can be considered as a study the possibilities to represent and recognize information in the style of creating and decoding puzzles.

Any programming paradigm can be supplemented with additional forms, such as declarativeness, abstractions, specification languages, etc., mainly solving problems such as “scaffolding,” that is, the aim of these forms is not an alternative or opposed representation of programming tools and methods, but temporary structure which used to support setting the boundaries of the behavior of programs, highlighting the processes that are convenient for practice.

5. Conclusions and Outlook

The Most long-lived programming languages support the practical paradigms of impertive-procedural, object-oriented, and functional programming. A growing number of languages support declarative, reflexive and meta-programming. New languages complement these paradigms with separate networking tools, leading to the formation of paradigms for remote access, parallel processes, supercomputer computing, and large data visualization. The complexity of multi-paradigm PLs can be overcome through the style of separate description of the paradigms supported in them [8].

The proposed methodology can be used to assess the complexity and complexity of programming, especially if supplemented by dividing the requirements for setting tasks in the fields of application into academic and industrial, and by the level of knowledge into clear, developed and complicated difficult to certify requirements.

Basic programming paradigms can be distinguished by ordering the main categories of semantic systems, and derivatives - by the difference between individual categories of semantic systems from the basic paradigm. Any programming paradigm can be enriched with additional paradigms for representing restrictive conditions for the functioning of programs. For this reason, they cannot be opposed to the actual PP. The classification of programming paradigms can depend on the degree of knowledge of the expanding space of tasks to be solved and the progress of available technical means that occurs within the framework of a stable class of tasks to increase efficiency.

The statements of the problems of parallel computing take into account that the speed of obtaining results on the available programs for solving a specific problem is insufficient. Paradigms of this direction are in the process of formation. Given the diversity of theoretical models in this area, it is natural to assume that there will be many such paradigms. There is reason to single out software aimed at providing feedback when working with devices and networks, on the surface-interface style of IT, and on supporting supercomputer processes.

In recent years, reasons have been discovered and understood for conditioning program verification by formalizing the programming paradigms used. Programming projects should be accompanied by a justification for the choice of not only software tools, but also paradigms in order to avoid inter-paradigm conflicts, fraught with subtle errors associated with changing and developing the functioning environment of long-lived programmable components. It is necessary to consider the difference between theoretical and practical programming, similar to the difference between elementary music theory and the performing skill of a musician.

DSL languages deserve special consideration as a new level of languages creation. If in ordinary PLs, the accumulation of programming experience is performed in the form of procedures, then DSL is the mechanism for accumulating experience in the form of languages.

The works of E. M. Lavrishcheva [7], Peter Van Roy [8] and Peter Wegner [6] should be mentioned as related works. E.M. Lavrishcheva presented a fairly complete overview of programming paradigms that is relevant for programming technologies [7], P. Van Roy analyzed more than 30 paradigms, mainly combined, and presented a diagram of their interconnections cited in Wikipedia [8], and P. Wegner performed a very serious analysis of OOP, methods for supporting this paradigm, and its comparison with other classical PPs [6].

References

1. https://www.levenez.com/lang/ — Computer Languages History.
2. http://progopedia.ru/ — Encyclopedia of programming languages (171 languages and 31 paradigms) *(In Russian)*.
3. Lavrov, S.S. Methods for definition the semantics of programming languages (Metody zadaniya semantiki yazykov programmirovaniya) // Programmirovaniye, 1978. № 6. S. 3–10 *(In Russian)*.
4. Floyd, R. W. (1979). The paradigms Programming .-- Communications of the ACM 22 (8): 455
5. Gorodnyaya L. V. On the presentation of the results of the analysis of languages and programming systems (O predstavlenii rezul'tatov analiza yazykov i sistem programmirovaniya) // Nauchnyy servis v seti Internet: trudy XX Vserossiyskoy nauchnoy konferentsii (17–22 sentyabrya 2018 g., g. Novorossiysk). M.: IPM im. M.V. Keldysha, 2018. https://keldysh.ru/abrau/2018/theses/46.pdf *(In Russian)*
6. Peter Wegner. Concepts and paradigms of object-oriented programming. SIGPLAN OOPS Mess. 1, 1 (August 1990), P. 7–87. https://pdfs.semanticscholar. DOI: http://dx.doi.org/10.1145/

7. Lavrishcheva E. M. Software engineering and technologies for programming complex systems. Textbook for universities. (Programmnaya inzheneriya i tekhnologii programmirovaniya slozhnykh sistem. Uchebnik dlya vuzov). M., 2018. 432 s. *(In Russian)*
8. Peter Van Roy. Classification of the principal programming paradigms. url: https://www.info.ucl.ac.be/~pvr/paradigms.html

1. https://www.info.ucl.ac.be/~pvr/paradigms.htmlhttps://www.info.ucl.ac.be/~pvr/paradigms.html

УДК 519.68, 681.3.06

Вопросы адаптивности в системах дистанционного обучения

Волянская Т.А. (Институт систем информатики СО РАН)

В статье рассматриваются вопросы реализации адаптивности в интеллектуальных системах дистанционного обучения, основанной на технологии адаптивной гипермедиа и моделях индивидуальных стилей обучения.

Ключевые слова: *интеллектуальные системы дистанционного обучения, адаптивные системы, архитектура интеллектуальных систем обучения, адаптивные обучающие гипермедиа-системы, адаптивная гипермедиа, адаптивное представление, адаптивная навигация, индивидуальные стили обучения, модель Колба, модель Хани-Мамфорда, модель Фельдера-Сильверман, модель когнитивных характеристик, таксономия Блума.*

1. Введение

В настоящее время вопросы дистанционного образования как никогда актуальны, интеллектуальные и адаптивные системы дистанционного обучения активно исследуются и развиваются. Появившиеся в последнее время адаптивные гипермедиа-системы существенно повышают возможности обучающих систем. Целью адаптивных систем является персонализация гипермедиа-системы, ее настройка на особенности индивидуальных пользователей [9,11].

Первые компьютерные обучающие системы (англ. Computer Assisted Instruction systems, CAI) появились в 1960-х годах. Они предлагали учащимся задания и запоминали их ответы. В 1970-х были разработаны системы, изменяющие представление учебного материала в зависимости от ответов учащихся. Это было началом моделирования обучаемых, но пока только их поведения, а не знаний [43].

В 1982 году Слимэн и Браун (D. Sleeman, J.S. Brown) впервые ввели термин интеллектуальных обучающих систем (Intelligent Tutoring Systems (ITS)) [37].

В 1990 годах появился новый термин – электронное (дистанционное) обучение (e-learning), система обучения при помощи информационных и электронных технологий, таких как интернет-обучение, компьютеризированное обучение, виртуальные классы и электронное сотрудничество, предоставляющие доступ к образовательному контенту с

использованием различных электронных средств информации. По сравнению с традиционным обучением, при котором преподаватель контролирует процесс преподавания и обучения, а также образовательный контент, электронное обучение ставит в центр самих учащихся, предоставляя им возможность учиться в интерактивном режиме, в своем собственном темпе, в простой, гибкой распределенной среде обучения [39].

Специальный класс дистанционных систем обучения – *интеллектуальные системы обучения (*англ. *Intelligent e-learning tutoring systems)* – являются результатом практического применения методов и средств искусственного интеллекта в области автоматизированного обучения и представляют собой новое поколение учебных систем. Интеллектуальные системы обучения предназначены улучшить процесс обучения в соответствии с индивидуальными характеристиками обучаемых, как и в случае традиционного репетиторства один на один. В процессе обучения в этих системах используются специальные знания трех основных типов: знания о предмете изучения, знания о методах обучения и знания об учащемся [41,44].

Предполагалось, что интеллектуальные системы обучения займут лидирующее место среди адаптивных систем обучения. Однако этого не произошло из-за их негибкости и высокой стоимости разработки. Для успешной реализации гибкого, динамичного, персонализированного учебного курса в интеллектуальных системах обучения необходимо определить архитектуру, обеспечивающую четкое различие между «механизмом адаптации», динамически генерирующим учебный курс, и «содержимым», используемым для генерирования этого персонализированного учебного курса [2,24].

Самая важная особенность репетиторства один на один состоит в том, что преподаватель способен адаптировать процесс обучения и преподавания к учащемуся, принимая во внимания его индивидуальные особенности. Не всегда возможно реализовать репетиторство в процессе обучения, однако, были предложены такие формы обучения, которые позволяют достичь приближенно равной эффективности, используя различные методы и средства адаптации. Методы и средства обучения, ориентированные на потребности отдельных учащихся, называют *адаптивным обучением* [1,9,11,12,22,29,30].

Характеристики учащихся, к которым может быть адаптировано обучение, называются *«предпочтениями»* или *«способностями» (aptitudes).* Проблеме соотношения уровня способностей учащихся и подбору соответствующих стратегий обучения посвящена отдельная область исследований, имеющая название *aptitude-*

treatment interaction (ATI), что можно перевести как «взаимодействие и взаимосвязь уровня способностей учащихся и стратегий обучения» [23].

Некоторые стратегии обучения более или менее эффективны для различных учащихся в зависимости от их индивидуальных способностей. ATI предполагает, что можно достичь оптимального результата в том случае, когда метод обучения точно соответствует способностям учащегося [38].

Индивидуальными стилями обучения принято называть способы, при помощи которых человек получает, обрабатывает и хранит информацию. Очевидно, что у учащихся бывают различные сильные стороны и предпочтения. Одни лучше работают с конкретной информацией, другие – с абстрактной теорией, одни лучше усваивают новое в виде рисунков, диаграмм, таблиц, другие – в виде словесных описаний. В общих чертах, теория стилей обучения основывается на том факте, что разным людям подходят различные стили обучения, и процесс обучения будет эффективным, если им будут даны подходящие инструкции, адаптированные к их стилю обучения [45].

Появившиеся в последнее время адаптивные гипермедиа-системы существенно повышают возможности обучающих систем. Целью адаптивных систем является персонализация гипермедиа-системы, ее настройка на особенности индивидуальных пользователей. Поддержка адаптивных методов в гипермедиа-системах оказывается весьма полезной в тех случаях, когда имеется одна система, обслуживающая множество пользователей с различными целями, уровнем знаний и опытом, и когда лежащее в ее основе гиперпространство относительно велико. Поэтому области применения адаптивной гипермедиа выходят далеко за границы обучающих систем и включают, например, такие, казалось бы, далекие от обучения области применения гипермедиа-систем, как открытые адаптивные виртуальные музеи [3-12,27-30].

В современных условиях в связи с развитием электронных информационно-образовательных ресурсов и сетевых технологий возрастает потребность разработки и создания адаптивных электронных учебных курсов в интеллектуальных системах дистанционного обучения [1,2].

В первом разделе статьи описываются интеллектуальные системы обучения и их архитектура, включающая модуль эксперта, модуль обучаемого, модуль преподавателя и модуль взаимодействия. Рассматриваются вопросы адаптивного обучения, ориентированного на потребности отдельных учащихся, вводится класс адаптивных систем обучения, в которых процесс обучения и проверки знаний адаптируется, основываясь на индивидуальных характеристиках учащихся, таких как априорные знания и предпочтения, стили обучения, когнитивные способности и т.п.

Второй раздел статьи посвящен адаптивным обучающим гипермедиа системам, реализующим адаптивное представление и адаптивную навигацию, основываясь на индивидуальном стиле обучения. Описываются два класса адаптации, применяемых в этих системах: адаптация на уровне содержания, называемая адаптивным представлением, которая включает методы и техники, адаптирующие содержание документа или стиль текста, и адаптация на уровне гиперссылок, называемая адаптивной навигационной поддержкой или адаптивной навигацией, которая адаптирует навигацию и ориентацию в гиперпространстве, изменяя вид и структуру гиперссылок. Приводятся различные методы и техники адаптивного представления и адаптивной навигации.

В третьем разделе статьи рассматривается теория стилей обучения, основанная на адаптации процесса обучения в соответствии с индивидуальными стилями обучения, под которыми понимаются различные способы получения, обработки и хранения информации учащимися. Описываются следующие модели, учитывающие индивидуальные стили обучения: модель Колба, модель Хани-Мамфорда, модель Фельдера-Сильверман, модель когнитивных характеристик и таксономия Блума [16,25,26,31,32].

2. Интеллектуальные системы обучения

Специальный класс дистанционных систем обучения – *интеллектуальные системы обучения (*англ. *intelligent e-learning tutoring systems)* – являются результатом практического применения методов и средств искусственного интеллекта в области автоматизированного обучения и представляют собой новое поколение учебных систем. Интеллектуальные системы обучения предназначены улучшить процесс обучения в соответствии с индивидуальными характеристиками обучаемых, как традиционное репетиторство один на один. В процессе обучения в этих системах используются специальные знания трех основных типов: знания о предмете изучения, знания о методах обучения и знания об учащемся [41,44].

2.1. Традиционная архитектура интеллектуальных систем обучения

Традиционная архитектура современных интеллектуальных систем обучения состоит из четырех компонентов: модуля эксперта, модуля обучаемого, модуля преподавателя и модуля взаимодействия [21].

Модуль эксперта (*Expert module*) выполняет две важные функции – служит источником знаний для учащихся и критерием для оценки успеваемости учащихся.

Модуль эксперта – центральный в ИСО, поскольку содержит базу знаний предметной области [13].

Модуль обучаемого (*Student module*) регистрирует понимание или непонимание учащимся знаний предметной области, т.е. содержит данные об успеваемости учащегося. Термин «модель обучаемого» (*Student model*) был впервые введен Слимэном и Брауном в 1982 г. для описания абстрактного представления учащихся [37].

Существует три основных типа моделей обучаемого [4]:

- оверлейная модель *(Overlay model)*, описывающая знания учащегося, как подмножество знаний эксперта;
- разностная модель *(Differential model)*, описывающая различия между знанием учащегося и знанием эксперта;
- пертурбационная модель *(Perturbation model)*, отражающая непонимание учащимся экспертных знаний.

Рис.1. Традиционная архитектура интеллектуальных систем обучения

Наиболее простой для реализации является *оверлейная модель*. Она строится в предположении, что знания обучаемого и знания системы имеют аналогичную структуру, при этом знания обучаемого являются подмножеством знаний системы. Каждому элементу знаний предметной области добавляется числовой атрибут, показывающий степень понимания учащимся материала по этой теме. Значение этого атрибута определятся в ходе опроса обучаемого.

Более сложная стратегия предложена для *разностной модели*. При построении этой модели система анализирует ответы учащегося и сравнивает их с теми знаниями, которые заложены в системе, и которыми пользуется эксперт при решении подобных задач. Различия между этими знаниями и ложатся в основу модели обучаемого. Эта модель позволяет учитывать не только отсутствие знаний у учащегося, но и неправильное их использование.

Пертурбационная модель строится в предположении, что знания учащегося и знания системы могут частично не совпадать. В этом случае важной предпосылкой построения такой модели является определение причин расхождений. Различают следующие причины расхождений:

1) недостаток знаний (учащийся не обладает знаниями, достаточными для того, чтобы правильно выполнить то или иное контрольное задание);

2) ошибочные знания (знания учащегося противоречат базе знаний системы);

3) неверное использование знаний (учащийся владеет необходимыми знаниями, но не умеет их правильно применять);

4) случайные ошибки (ошибки в вычислениях или ошибки, порожденные невнимательным чтением формулировок заданий и вариантов ответов);

5) умышленные ошибки (ошибки, возникающие, когда учащийся использует какую-либо «стратегию» ответа на контрольные задания, например, всегда выбирает только первый вариант ответа).

Модуль преподавателя (Tutor module) тесно связан с модулем обучаемого и принимает решения о том, как обучать каждого учащегося индивидуально. Модуль преподавателя должен выполнять три функции: (1) контролировать выбор и порядок предоставления учебных материалов, (2) отвечать на вопросы учащегося, (3) распознавать, когда и какая помощь нужна учащемуся. Эти задачи, решаемые модулем преподавателя, называются *сценариями обучения* [36].

Модуль взаимодействия (Communication module) контролирует взаимодействие между системой и обучаемым. Взаимодействие может быть реализовано через диалоговый и графический интерфейс пользователя [45].

2.2. Адаптивные системы

Адаптивные системы (Adaptive Systems, AS) – системы, которые могут изменять свою структуру, функциональность или интерфейс таким образом, чтобы приспосабливаться к различным, изменяющимся со временем, потребностям отдельного пользователя или групп пользователей [15].

Различают *адаптируемые (Adaptable)* системы, которые позволяют пользователю (обучаемому) изменять некоторые параметры и соответственно адаптировать свое поведение, и *адаптивные (Adaptive)* системы, которые автоматически адаптируются к пользователю, основываясь на своих предположениях о нем [6,35].

Адаптивные системы и дистанционное обучение определяют новый класс систем – *адаптивные системы дистанционного обучения (Adaptive E-learning Systems),* которые изменяют процесс изучения, обучения и проверки знаний учащегося, основываясь на его индивидуальных характеристиках [19,42]. Таким образом, они адаптируют выбор и представление учебного материала в соответствии со знаниями, потребностями, когнитивными характеристиками, стилями обучения, априорными знаниями и предпочтениями учащегося. Иногда они называются также *адаптивными системами обучения (Adaptive Educational Systems, AES)* [17].

Два наиболее известных класса адаптивных систем дистанционного обучения, это *интеллектуальные системы обучения (Intelligent Tutoring Systems, ITS),* которые адаптируются к характеристикам учащихся, ориентированных на знания (цели, знания, опыт), и *адаптивные обучающие гипермедиа-системы (Adaptive Educational Hypermedia Systems, AEHS),* которые адаптируются в основном к стилям обучения.

Существует два основных подхода для реализации адаптивности: макро-адаптация и микро-адаптация [40].

Макро-адаптация происходит перед процессом обучения: вначале собираются данные о когнитивных способностях учащегося, а затем они используются для принятия решения о выборе типа среды обучения и метода обучения, наиболее подходящих при таких способностях.

Напротив, *микро-адаптация* происходит во время процесса обучения. Она заключается в изменении содержания, а не способа представления. Эти решения основываются на сравнении текущих знаний учащихся со знаниями, которые они должны иметь после окончания процесса обучения. Когда учащиеся демонстрируют недостаток знаний, новая или предшествующая информация представляется повторно, а в том случае, если учащиеся демонстрируют хорошее знание, разрешается пропуск некоторых частей.

Некоторые стратегии обучения в разной степени эффективны для различных учащихся в зависимости от их индивидуальных способностей. ATI предполагает, что можно достичь оптимального результата в случае, когда метод обучения точно соответствует способностям учащегося [38].

ATI модель для генерирования адаптивного учебного курса включает восемь этапов:

1) идентифицировать цели,

2) специфицировать характеристики задачи,

3) идентифицировать начальное множество характеристик учащихся,

4) выбрать самые важные характеристики учащихся,

5) проанализировать учащихся в целевой группе,

6) выбрать желательное изменение (в успеваемости учащихся),

7) определить, как адаптировать обучение,

8) разработать альтернативные методы обучения.

Существует много различных характеристик учащихся, к которым адаптируются системы обучения, такие как знания, стили обучения, когнитивные характеристики, предпочтения и т.д. Два более важных подхода в реализации адаптивности в системах обучения, это адаптация к *стилям обучения* и адаптация к *когнитивным характеристикам* [33].

3. Адаптивные обучающие гипермедиа-системы

Адаптивные системы, реализующие стили обучения в процессе обучения, называются *адаптивными обучающими гипермедиа-системами (АОГС)*. Они адаптируют только представление к стилю обучения учащегося, но не используют эту информацию для решения о том, что представить учащемуся [14,20].

Адаптивные обучающие гипермедиа-системы используют адаптивное представление и адаптивную навигацию, основываясь на индивидуальном стиле обучения [18,20].

На некотором уровне обобщения гипермедиа-системы состоят из набора узлов или гипердокументов (веб-страниц), связанных гиперссылками. Каждая страница содержит некоторую локальную информацию и ряд ссылок на связанные страницы. Кроме того, гипермедиа-системы могут включать индекс и глобальную карту, которые обеспечивают ссылки на все доступные страницы. Все, что может быть адаптировано в адаптивной гипермедиа, – это содержание обычных страниц (адаптация на уровне содержания) и гиперссылки с обычных страниц, страниц индексов и карт (адаптация на уровне гиперссылок). Адаптация на уровне содержания и гиперссылок рассматриваются как два различных класса адаптации гипермедиа. Первый называется *адаптивным представлением (Adaptive presentation)*, а второй – *адаптивной навигационной поддержкой* или *адаптивной навигацией (Adaptive navigation support)* [1,6].

Адаптивное представление адаптирует содержание документа или стиль текста. Адаптивная навигация обеспечивает учащимся ориентацию и навигацию в гиперпространстве, изменяя внешний вид контента, доступного по гиперссылкам.

3.1. Адаптивное представление

Смысл методов адаптивного представления состоит в том, чтобы адаптировать содержание страницы, к которой обращается учащийся, к текущему знанию, целям и другим его характеристикам. Например, начинающему ученику могут потребоваться дополнительные объяснения, в то время как продолжающему можно предоставить более детальную и сложную информацию. Адаптивное представление – общий термин для всех методов и техник, которые адаптируют содержание гипермедиа-страницы в соответствии с моделью обучаемого.

3.1.1. Методы адаптивного представления

Содержание гипермедиа документа адаптируется к потребностям учащегося посредством сокрытия некоторой специализированной информации или добавления дополнительной информации. Основные методы адаптивного представления текста: дополнительные, предварительные и сравнительные объяснения, варианты объяснений и сортировка.

Дополнительные объяснения (additional explanations). Цель метода дополнительных объяснений, наиболее популярного метода адаптации содержания – показать учащемуся те части информации об отдельном понятии, которые соответствуют его знаниям или целям, и скрыть те, которые не соответствуют.

Например, подробности нижнего уровня могут быть скрыты от учащихся с недостаточным уровнем знания этой темы, поскольку они не смогут их понять. Напротив, дополнительные пояснения, обычно требующиеся начинающим для понимания темы, могут быть скрыты от учащихся с хорошим уровнем знания этой темы, поскольку они больше в них не нуждаются. В общих чертах, вдобавок к основному представлению, некоторая категория учащихся может получить дополнительную информацию, которая специально подготовлена для этой категории, и не будет показываться учащимся других категорий.

Предварительные объяснения (prerequisite explanations) и сравнительные объяснения (comparative explanations). Два других метода – предварительные (необходимые как условия) и сравнительные объяснения – изменяют представляемую о

понятии информацию в зависимости от уровня знаний учащимся сходных (близких) понятий.

Первый метод основан на предварительных связях между понятиями. Идея в следующем: перед предоставлением объяснения понятия система добавляет объяснения всех понятий, предварительно необходимых для понимания этого понятия, которые еще не известны учащемуся.

Второй метод основан на отношении подобия (сходства) между понятиями. Смысл метода состоит в том, чтобы при объяснении нового понятия подчеркивать его связь с уже известными понятиями. Если понятие, сходное с представляемым понятием, уже известно, учащийся получает сравнительное объяснение, которое подчеркивает сходства и различия между текущим и сходным понятиями.

Варианты объяснения (explanation variants). Очевидно, что для адаптации не всегда достаточно показа или сокрытия некоторых частей информации, поскольку различные учащиеся могут нуждаться в существенно различной информации. При использовании метода вариантов объяснения система хранит несколько различных вариантов объяснения отдельного понятия, и учащийся получает вариант, наиболее соответствующий своей модели.

Сортировка (sorting). Метод, который может принимать во внимание как подготовку, так и уровень знаний учащегося – сортировка фрагментов информации о понятии. Фрагменты информации сортируются и отображаются на странице от наиболее до наименее релевантного подготовке и знанию учащегося.

3.1.2. Техники адаптивного представления

Следующие техники используются для реализации перечисленных выше методов адаптивного представления текста.

Условный текст (conditional text). Простая, но эффективная техника для адаптации содержания – технология условного текста. При использовании этой техники вся возможная информация о понятии разделена на несколько частей текста. Каждая часть связана с условием на уровне знания, представленного в модели обучаемого. При представлении информации о понятии система показывает только те части текста, для которых условие истинно. Это техника нижнего уровня, требующая некоторой работы «программирования» от автора для установления всех требуемых условий. Выбирая соответствующие условия на уровне знаний текущего понятия и связанных понятий, представленных в модели обучаемого, можно реализовать все методы адаптации, перечисленные выше, за исключением сортировки. Простой пример – это сокрытие

частей текста с неподходящими объяснениями, если уровень знаний учащегося текущего понятия достаточно хорош, или включение части текста со сравнительными объяснениями, если соответствующее сходное понятие уже известно.

Расширяющийся текст (stretchtext). Техника более высокого уровня, которая дополнительно может включать и выключать различные части текста в соответствии с уровнем знаний учащихся. Она основана на так называемом «расширяющемся» тексте (stretchtext), который является специальным видом гипертекста. В обычном гипертексте результатом активации ключевого слова является перемещение на другую страницу со связанным текстом. В stretchtext этот связанный текст может просто заменять активированное слово (или фразу с этим словом), расширяя текст текущей страницы. При необходимости этот расширенный, или «развернутый» текст может быть свернут обратно в слово. Каждый узел – stretchtext-страница, которая может содержать много «свернутых» слов. Идея адаптивного stretchtext-представления состоит в том, чтобы представить требуемую страницу со всеми stretchtext-расширениями, нерелевантными для учащегося, свернутыми, и всеми расширениями, релевантными для учащегося – развернутыми.

Важная особенность адаптивной stretchtext-техники состоит в том, что она позволяет и учащемуся, и системе адаптировать содержание отдельной страницы, и в том, что она может принимать во внимание и знания, и предпочтения учащегося. После произвольного представления stretchtext-страницы, она может быть далее адаптирована учащимся, который может сворачивать или разворачивать соответствующие объяснения и подробности в соответствии со своими предпочтениями. Модель обучаемого может обновляться в соответствии с демонстрируемыми им предпочтениями, чтобы гарантировать, что он всегда будет видеть предпочитаемую комбинацию сокращенных и несокращенных частей. Например, если учащийся свернул дополнительные объяснения отдельного понятия, они будут показываться свернутыми до тех пор, пока он не изменит предпочтения.

Варианты фрагмента (fragment variants) и варианты страницы (page variants). Метод вариантов объяснения может быть реализован с помощью техник вариантов фрагмента и вариантов страницы. Варианты фрагмента – более мелкомодульная реализация метода вариантов объяснения. Система хранит несколько вариантов объяснения каждого понятия (варианты фрагментов), и учащийся получает страницу, содержащую те варианты, которые соответствуют его знанию о представленных понятиях. Техники вариантов страницы и вариантов фрагмента могут быть скомбинированы, например, для обеспечения адаптации и к подготовке, и к знанию

учащегося. Текущий вариант страницы для конкретного узла выбирается согласно подготовке учащегося. Эта страница затем может быть адаптирована: для каждого понятия, представленного на странице, система выбирает объяснение, которое наиболее соответствует уровню знаний учащегося.

Варианты страницы – наиболее простая техника адаптивного представления. При использовании этой техники система хранит несколько вариантов страницы с различными представлениями ее содержания. Как правило, каждый вариант подготовлен для одного из возможных стереотипов учащихся. При представлении страницы система выбирает вариант страницы согласно стереотипу учащегося.

Фреймовая техника (frame-based technique). Наиболее мощная из всех техник адаптации содержания – фреймовая (основанная на фреймовом представлении) техника. При использовании этой техники вся информация об отдельном понятии представлена в форме фрейма. Слоты фрейма могут содержать несколько вариантов объяснения понятия, связи с другими фреймами, примерами и т.д. При использовании технологии естественного языка страницы монтируются из маленьких информационных элементов подобно словам и частям предложений. Используются специальные правила представления для определения того, какие слоты должны быть представлены конкретному учащемуся и в каком порядке. Более точно, эти правила используются для выбора одной из существующих схем представления (каждая схема – упорядоченное подмножество слотов), и схема используется для представления понятия. Могут применяться правила для вычисления «приоритета представления» для каждого слота, и затем подмножество слотов с высоким приоритетом представляется в порядке уменьшения приоритета. В своих условных частях эти правила могут ссылаться не только на уровень знаний пользователем представляемого понятия, но и на любую характеристику, представленную в модели пользователя. В частности, система может принимать во внимание подготовку учащегося.

Фреймовая техника может использоваться для реализации всех методов, упомянутых выше. Методы предварительных и сравнительных объяснений могут быть реализованы при помощи фреймовой техники, при этом соответствующие условия ставятся на уровне знаний связанных понятий.

3.2. Адаптивная навигация

Смысл методов адаптивной навигационной поддержки состоит в том, чтобы помочь учащимся найти путь в гиперпространстве с помощью адаптации способа представления гиперссылок к целям, знаниям и другим характеристикам отдельного

учащегося. Эти методы могут быть классифицированы согласно способу, который они используют для адаптации представления ссылок. Различаются шесть методов для адаптации представления ссылок: полное руководство (*direct guidance*), сортировка ссылок (*link sorting*), сокрытие ссылок (*link hiding*), аннотирование ссылок (*link annotation*), генерирование ссылок (*link generation*) и адаптация карты (*map adaptation*) [1,6,17,18].

3.2.1. Методы адаптивной навигационной поддержки

Техники адаптивной навигационной поддержки используются для достижения нескольких целей адаптации: обеспечить глобальное руководство, локальное руководство, локальную ориентацию, глобальную ориентацию и управление индивидуализированными представлениями в информационных пространствах.

Глобальное руководство (global guidance). Глобальное руководство можно обеспечить в гипермедиа-системах, в которых учащиеся имеют некоторую «глобальную» информационную цель (т. е. нуждаются в информации, которая содержится в одном или нескольких узлах где-нибудь в гиперпространстве) и просмотр – способ нахождения требуемой информации. Цель методов глобального руководства состоит в том, чтобы помочь учащемуся найти самый короткий путь к информационной цели с минимальным блужданием.

Наиболее полный метод обеспечения глобального руководства состоит в том, чтобы на каждом шаге просмотра предлагать учащемуся ссылку для дальнейшего перехода (т. е. применять технику полного руководства). Более поддерживающий метод состоит в применении техники адаптивной сортировки: все гиперссылки с данной страницы сортируются в соответствии с релевантностью глобальной цели (наиболее релевантные – сначала). При этом учащиеся все еще имеют возможность перехода по первой наиболее релевантной ссылке, но также имеют некоторую информацию (релевантность других ссылок), чтобы сделать свободный выбор.

Локальное руководство (local guidance). Цель методов локального руководства состоит в том, чтобы помочь учащемуся сделать один навигационный шаг, предлагая гиперссылки с текущей страницы, наиболее подходящие для перехода. Эта цель отчасти подобна цели глобального руководства, но более «скромна». Методы локального руководства не предполагают глобальную цель для обеспечения руководства. Они делают указание согласно предпочтениям, знанию и подготовке пользователя – что наиболее важно для данной прикладной области.

Методы, используемые в гипермедиа-системах обучения – это сортировка ссылок согласно знанию учащегося и полное руководство в соответствии со знанием учащегося. Последний метод обычно применяется для выбора наиболее подходящей задачи из набора задач, доступных из текущего пункта.

Поддержка локальной ориентации (local orientation support). Цель методов поддержки локальной ориентации состоит в том, чтобы помочь учащемуся в локальной ориентации (т. е. помочь понять, что находится вокруг, и какова его относительная позиция в гиперпространстве). Существующие системы осуществляют поддержку локальной ориентации двумя различными способами: предоставляя дополнительную информацию о страницах, доступных с текущей (использование техники аннотирования), и ограничивая число навигационных возможностей для уменьшения познавательной перегрузки, позволив учащимся сфокусироваться на анализе наиболее релевантных гиперссылок (использование техники сокрытия).

Методы, основанные на технике сокрытия, имеют ту же самую идею: скрыть от учащегося все гиперссылки (или с индекса, или с локальной страницы), которые не подходят ему в данный момент, или, другими словами, показывать только релевантные гиперссылки (соответствующие знанию, цели, опыту или предпочтениям учащегося). Другой метод, основанный на опыте учащегося в данном гиперпространстве, состоит в том, чтобы показывать учащимся тем больше ссылок, чем больше опыта они имеют в гиперпространстве.

В гипермедиа-системах обучения применяются два специфичных метода, основанных на технике сокрытия: смысл первого метода заключается в сокрытии гиперссылок к тем частям информации, которые учащийся не готов еще изучить, а второго – к тем, которые относится к целям обучения последующих уроков.

Цель применения методов, основанных на технике аннотирования, состоит в том, чтобы дать учащемуся информацию о текущем «состоянии» документов за видимыми гиперссылками. Аннотирование может использоваться, чтобы показать несколько градаций релевантности гиперссылки или несколько уровней знания учащимся информации, содержащейся по аннотируемым гиперссылкам, а также чтобы выделить гиперссылки, соответствующие текущей цели, и затемнить несоответствующие.

Методы поддержки локальной ориентации не руководят учащимся непосредственно, но обеспечивают помощь в понимании того, что содержится за ближайшими ссылками, а также в принятии обоснованного навигационного выбора.

Поддержка глобальной ориентации (global orientation support). Цель применения методов поддержки глобальной ориентации состоит в том, чтобы помочь учащемуся

понять структуру всего гиперпространства и свою абсолютную позицию в нем. В неадаптивной гипермедиа эта цель обычно достигается с помощью визуальных ориентиров и глобальных карт. Адаптивные гипермедиа могут обеспечить более существенную поддержку для учащегося благодаря техникам аннотирования и сокрытия.

Аннотации играют роль ориентиров: так как гиперссылка сохраняет свою аннотацию при просмотре из различных мест гиперпространства: учащийся легко может узнавать гиперссылки, которые он встречал прежде, и понимать свою текущую позицию. Сокрытие уменьшает размер видимого гиперпространства и может упростить и ориентацию, и обучение, обеспечивая постепенное изучение гиперпространства. Перспективное направление применения адаптивной поддержки глобальной ориентации – адаптация локальных и глобальных карт, когда сама структура карты может зависеть от характеристик учащегося.

Управление индивидуализированными представлениями (managing personalized views). Индивидуализированные представления – способ организации электронного рабочего места для учащихся, которые нуждаются в доступе к достаточно небольшой части гиперпространства для своего обучения. Каждое представление – это просто список гиперссылок ко всем документам, которые соответствуют конкретной учебной цели. Классические гипермедиа-системы предлагают закладки и списки как способы создавать персональные представления. Более развитые системы предлагают механизмы адаптируемости более высокого уровня, основанные на метафорах и моделях обучаемого.

Адаптивные решения, т. е. управляемая системой поддержка индивидуализированных представлений, требуются в WWW-подобных динамических информационных пространствах, где объекты могут появляться, исчезать или изменяться. Для этого используются интеллектуальные агенты, которые могут регулярно искать новые релевантные объекты и распознавать измененные объекты или объекты с истекшим сроком хранения.

3.2.2. Техники адаптивной навигационной поддержки

Прямое руководство (direct guidance). Прямое руководство – наиболее простая техника адаптивной навигационной поддержки. Прямое руководство может применяться в любой системе, которая может определить «следующую наилучшую» страницу для учащегося в соответствии с его целью и другими параметрами,

представленными в модели обучаемого. Гиперссылка предоставляется к той странице, которую система считает наиболее подходящей для следующего перехода учащегося.

Различаются два метода прямого руководства:

- «следующий наилучший» (*next best*) – предоставление кнопки «next» для навигации через гиперпространство,
- установление последовательности страниц (page sequencing or trails) – генерирование последовательности страниц для просмотра всей гипермедиа-системы или ее части.

Прямое руководство может использоваться со всеми видами гиперссылок, но оно вряд ли может быть основной формой навигационной поддержки, поскольку не обеспечивает никакой помощи для учащихся, которые не хотели бы следовать указаниям системы.

Адаптивная сортировка (упорядочение) ссылок (adaptive sorting (ordering) of links). Вместо предоставления единственной «наилучшей» гиперссылки, эта техника предоставляет список гиперссылок, упорядоченных по убыванию релевантности, в соответствии с моделью обучаемого. Различаются два метода сортировки:

- сортировка по подобию (similarity sorting) – ссылки сортируются, исходя из их подобия данной странице;
- предварительные знания (prerequisite knowledge) – ссылки упорядочиваются согласно предварительно необходимым знаниям.

Адаптивное упорядочение имеет ограниченную применимость: эта техника может использоваться только с контекстно-независимыми гиперссылками. Другая проблема с адаптивным упорядочением состоит в том, что эта техника делает порядок гиперссылок неустойчивым: он может изменяться каждый раз при входе учащегося на страницу, в то время как устойчивый порядок элементов меню важен для начинающих.

Адаптивное сокрытие ссылок (adaptive link hiding). Адаптивное сокрытие ссылок – в настоящее время наиболее часто используемая техника для обеспечения адаптивной навигационной поддержки. Идея заключается в том, чтобы ограничить навигационное пространство, скрывая гиперссылки к «нерелевантным» страницам. Страница может рассматриваться как нерелевантная по нескольким причинам: например, если она не связана с текущей целью учащегося, или если она представляет материал, который он не готов еще понять. Сокрытие защищает учащихся от сложности неограниченного гиперпространства и уменьшает их познавательную перегрузку, также оно более понятно и выглядит более «стабильно», чем адаптивное упорядочение.

Различаются три вида сокрытия гиперссылок: непосредственно сокрытие гиперссылок (ссылка делается неотличимой от нормального текста), удаление гиперссылок (ссылка делается невидимой) и отключение гиперссылок (ссылка делается недоступной):

Адаптивное сокрытие ссылок (adaptive link hiding): эта техника делает гиперссылку визуально неотличимой от обычного текста. Сокрытие имеет широкую применимость: оно может использоваться со всеми видами контекстно-независимых, индексных ссылок и ссылками карт с помощью реального сокрытия кнопок или пунктов меню и с контекстно-зависимыми ссылками, превращая ключевые слова в обычный текст.

Адаптивное удаление ссылок (adaptive link removal): эта техника полностью удаляет гиперссылку для нерелевантных ссылок. Это может быть выполнено только тогда, когда окружающий текст все еще имеет смысл после удаления гиперссылки.

Адаптивное отключение ссылок (adaptive link disabling): функциональные возможности гиперссылки удаляются, но текст ссылки остается. Эта техника часто объединяется с сокрытием гиперссылки, потому что, если визуальное представление гиперссылки остается, но ссылка не «работает», учащийся может подумать, что это ошибка в системе.

Адаптивное аннотирование ссылок (adaptive link annotation). Смысл техники адаптивного аннотирования ссылок состоит в пополнении ссылок некоторыми комментариями, чтобы дать учащимся подсказку о содержании страниц по аннотируемым ссылкам. Аннотации могут быть представлены в текстовой форме или в форме визуальных подсказок, например, используя различные значки, цвета или размеры шрифтов.

Наиболее популярный способ аннотирования ссылок – использование метафоры светофора (*traffic light metaphor*). Гиперссылка аннотируется цветной точкой: красная точка перед ссылкой указывает, что у учащегося недостаточно знаний для понимания этой страницы; таким образом, страница не рекомендуется для чтения. Желтая точка означает, что страница не рекомендуется для чтения; эта рекомендация менее строга, чем в случае красной точки. Зеленая точка ставится перед ссылками на страницы, рекомендуемыми для чтения. Серая точка указывает учащемуся, что содержание этой страницы уже известно. Существуют и другие варианты.

Типичный вид аннотирования, рассматриваемый в традиционной гипермедиа – статическое (независимое от учащегося) аннотирование.

Аннотирование может использоваться со всеми возможными типами гиперссылок. Эта техника сохраняет стабильный порядок гиперссылок и избегает проблем с

формированием неправильных ментальных карт. Аннотирование – более мощная техника, чем сокрытие: сокрытие может различать только два состояния – уместные и неуместные, в то время как аннотирование – от шести состояний, в частности, несколько уровней релевантности. Эксперименты привели к заключению, что адаптивное аннотирование гиперссылок является полезным для сокращения числа навигационных шагов и в улучшении понимания учебного материала.

Адаптивное генерирование ссылок (adaptive link generation). Рост систем рекомендаций делает необходимым различать два по существу различных способа адаптивной навигационной поддержки: адаптация гиперссылок, представленных на странице во время разработки гиперпространства, и генерирование новых (неразработанных) гиперссылок для страницы. Генерирование гиперссылок включает три случая: обнаружение новых полезных ссылок между документами и добавление их к постоянному набору существующих ссылок, генерирование ссылок для навигации между элементами, основанной на подобии, и динамическая рекомендация релевантных ссылок.

Адаптация карты (map adaptation). Техника адаптации карты включает различные пути адаптации формы глобальных и локальных гипермедиа карт (графические представления структуры гиперссылок), представленных обучаемому. Карты могут быть адаптивно фильтрованы, чтобы обеспечить представление частей гипердокумента, которые релевантны для учащегося. Такие техники, как полное руководство, сокрытие и аннотирование, могут использоваться для адаптации гипермедиа-карты, но все эти техники не изменяют форму или структуру карт.

Полное руководство, сортировка, сокрытие, аннотирование и адаптация карты – основные техники адаптивной навигационной поддержки. Большинство из существующих методов адаптации используют только один из этих способов для обеспечения адаптивной навигационной поддержки. Однако эти техники не противоречивы и могут использоваться в комбинациях. В частности, техника полного руководства может легко использоваться в комбинации с любой из трех других технологий [6].

При рассмотрении адаптации в системах обучения необходимо различать, какие системные компоненты должны быть адаптированы, и на основе чего (когда, как и зачем) [34]:

1. Адаптационная информация определяет то, какую форму адаптации применить. Обычно это знания учащегося, однако, адаптация может базироваться на любом состоянии системного окружения.

2. Правила адаптации необходимы для принятия решений о том, когда начать процесс адаптации. Эти правила базируются на адаптационной информации и считаются пусковыми механизмами для адаптации (т.н. триггерами).

3. Процедуры адаптации показывают, какие компоненты системы требуют адаптации и каким образом.

4. Цели адаптации относятся к задачам адаптации, но не к ресурсам. По этой причине, компоненты, моделирующие цели адаптации, должны также отслеживать состояния окружения и оценивать эффект адаптации.

Методы адаптации (*adaptation methods*) – это методы, определенные на концептуальном уровне, тогда как техники (или технологии) адаптации (a*daptation techniques*) – это реализации методов адаптации. При определении того, как адаптировать, необходимо принять во внимание следующее: где (область применения адаптивной системы), зачем (каковы цели адаптации), к чему (к каким характеристикам адаптируется система), что (на содержании или навигации сосредоточена адаптация), как (какие методы и техники адаптации используются) [18].

Адаптивное представление содержания документов или стиля текста реализовано в гипермедиа-системах обучения C-Book и Hypertext. В системах ELM-ART, InterBook, WEST-KBNS, AST поддерживается адаптивная навигация, обеспечивающая учащимся ориентацию и навигацию в гиперпространстве посредством изменения внешнего вида видимых гиперссылок [17].

WAPE – адаптивная среда дистанционного обучения, поддерживающая активное индивидуальное обучение программированию в рамках проблемного подхода и соединяющая возможности адаптивных гипермедиа-систем и интеллектуальных обучающих систем [9,11,12,29,30].

Система WAPE ориентирована на поддержку дистанционного обучения и предполагает четыре типа пользователей: студенты, инструкторы, лекторы и администраторы. Все пользователи осуществляют доступ к системе через стандартный Web-браузер, который представляет HTML-документы, предоставляемые HTML-сервером на стороне сервера.

Многие адаптивные системы отслеживают движение пользователя по гиперкниге. Хотя это оправданный подход, его недостатком является трудность измерения знания, приобретенного учащимся во время чтения Web-страницы. Вместо этого в системе WAPE для обновления модели знаний используются только данные об успехах и неудачах студента, продемонстрированных им при решении задач (тестов, заданий и упражнений).

Другое важное отличие подхода от традиционного состоит в том, что в системе WAPE успех студента в изучении курса не оценивается на основании уровня знаний в его модели. Поэтому достижение определенного уровня знаний студентом не позволяет ему завершить изучение курса с определенной оценкой, а лишь дает возможность приступить к решению той или иной задачи из его индивидуального набора. Все это мотивировано проблемным подходом к обучению, который поддерживается системой WAPE.

4. Теория стилей обучения

Индивидуальными стилями обучения принято называть способы, при помощи которых человек получает, обрабатывает и хранит информацию. Очевидно, что учащиеся имеют различные сильные стороны и предпочтения. Одни лучше работают с конкретной информацией, другие – с абстрактной теорией, одни лучше усваивают новое в виде рисунков, диаграмм, таблиц, другие – в виде словесных описаний. В общих чертах, теория стилей обучения основывается на том факте, что разным людям подходят различные стили обучения, и процесс обучения будет эффективным, если им будут даны подходящие инструкции, адаптированные к их стилю обучения [1,45].

4.1.Модель Колба

Заслуга первых исследований в области стилей обучения принадлежит американскому ученому Дэвиду Колбу (David Kolb), который в 1984 году построил циклическую четырехступенчатую *эмпирическую модель процесса обучения* и усвоения человеком новой информации *(Kolb's Experiential Learning Model)* [1,32].

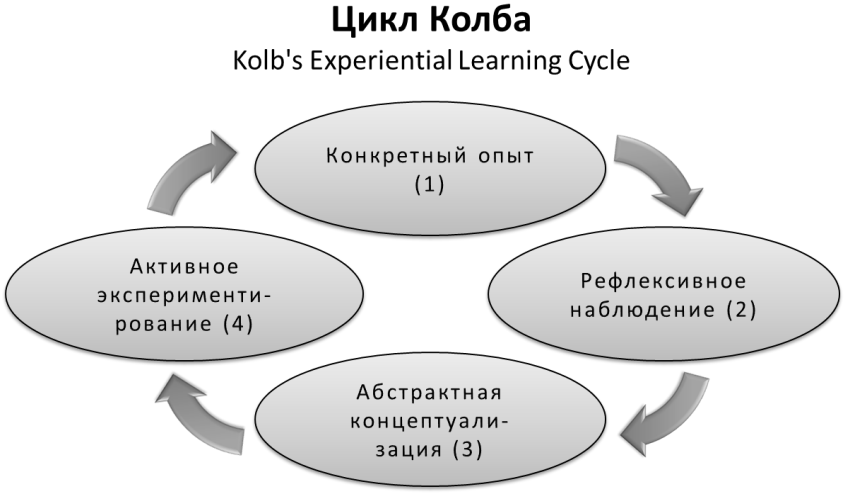


Рис. 2. Цикл Колба

Первая ступень, согласно Колбу, – это приобретение непосредственного, *конкретного опыта (Concrete Experience)*, который дает материал для второй ступени, *рефлексивного наблюдения (Reflective Observation),* в ходе которого учащийся обдумывает, что он только что узнал. Затем следует третья ступень, *абстрактная концептуализация (Abstract Conceptualization)* – это стадия теоретического обобщения, осмысления новых знаний, добавления новой информации и установления взаимосвязей. Четвертая стадия, *активного экспериментирования (Active Experimentation)* – это экспериментальная проверка новых знаний и самостоятельное применение на практике, перевод знаний в умения и навыки.

Процесс обучения может начинаться с любой стадии. Он протекает циклически – до тех пор, пока не сформируется требуемый навык.

Теория *циклов обучения (Kolb's Experiential Learning Cycle)* и *стилей познания Колба (Kolb's Learning Styles Model)* основывается на том, что при получении информации учащийся обращает внимание и усваивает одни виды информации в большей степени, чем другие. Подобные стратегии информационного выделения и игнорирования образуют характерный стиль восприятия и обработки информации, называемый *стилем познания*.

Рис.3. Эмпирическая модель обучения Колба

Стиль познания в теории Колба имеет два основных измерения: способ восприятия и сбора информации (конкретный опыт противопоставляется здесь абстрактной концептуализации) и способ реакции и обработки информации (рефлексивное наблюдение противопоставляется здесь активному экспериментированию).

Что касается способа сбора информации, некоторые учащиеся предпочитают получать информацию посредством конкретного, реального опыта. Оптимальным условием для их обучения является их непосредственное участие в опыте. Другие, напротив, лучше усваивают абстрактную, символическую или теоретическую информацию. Наилучшим условием для их обучения является представление им идей и теорий и возможности их логического анализа и осмысления. Такие учащиеся склонны к абстрактной концептуализации. Учащиеся, склонные к конкретике, предпочитают взаимодействовать с людьми, а абстрактные концептуалисты получать информацию из книг.

Второе измерение относится к стратегиям интерпретации и оценки информации, а также реакции на нее. К примеру, некоторые учащиеся, получив какую-то информацию, начинают рассматривать ее с различных сторон, размышлять и анализировать. Они склонны к рефлексивному наблюдению. Для них характерно избегание быстрых решений, стремление взвешивать и только после этого реагировать на информацию. Напротив, учащиеся, склонные к активному экспериментированию, реагируют на информацию едва ли не мгновенно. Они оценивают новую информацию опытным путем, пытаясь использовать ее на практике в конкретных ситуациях и для решения конкретных проблем, формулируют альтернативные гипотезы относительно направлений применения.

Таким образом, теория Колба определяет четыре различных стиля обучения: *аккомодационный, дивергентный, ассимиляционный и конвергентный*. Каждый стиль обучения представляет собой комбинацию двух предпочитаемых способов получения и обработки информации.

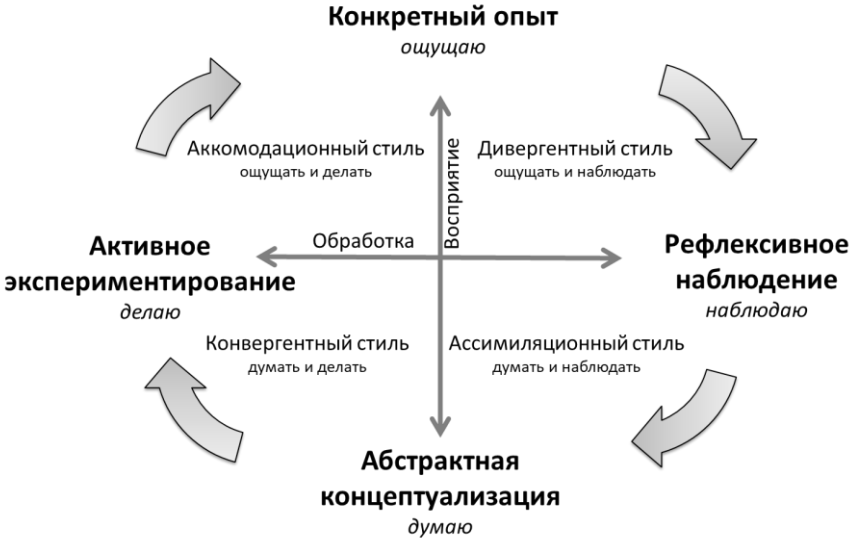


Рис.4. Модель стилей обучения Колба

Для *аккомодационного стиля (Accommodating learning style)* познания характерно получение информации посредством конкретного опыта и обработка информации посредством активного экспериментирования. Люди с этим стилем познания способны обучаться на конкретном опыте. Они четко планируют свою деятельность и любят экспериментировать с чем-то новым и содержащим вызов. Оптимальная сфера деятельности для них – сфера практического жизненного опыта. Они предпочитают заниматься участием в реальных проектах, постановкой целей, распределением заданий, а также опытной проверкой различных подходов к решению проблемы. В обучении предпочитают совместно выполнять задания, достигать целей, решать практические задачи.

Для *дивергентного стиля познания (Diverging learning style)* характерно получение информации посредством конкретного опыта и обработка информации посредством рефлексивного наблюдения. Люди с подобным стилем познания уверенно чувствуют себя в ситуациях, требующих генерации новых идей и выработки альтернативных перспектив. Они предпочитают заниматься творческой деятельностью, поиском всевозможной информации и проведением «мозговых штурмов». Отличаются развитым воображением и необычайной широтой интересов, эмоциональностью и стремлением к работе в группах.

Для *ассимиляционного стиля (Assimilating learning style)* характерно рефлексивное наблюдение и абстрактная концептуализация. Люди с таким стилем познания предпочитают работать с абстрактными идеями и концепциями, широко применяя методы индукции. Они способны к обработке больших объемов информации и изложения ее в точной, компактной и логичной форме, создании теоретических моделей. Они предпочитают работать в сфере науки, заниматься деятельностью, основным элементом которой является поиск и анализ информации.

Для *конвергентного стиля (Converging learning style)* характерна абстрактная концептуализация и активное экспериментирование. Лица, которым присущ этот стиль, склонны к экспериментированию с новыми идеями и сильны в применении на практике идей и теорий. Они предпочитают работать в инженерной и технологической сферах, заниматься практическими приложениями результатов исследований, решением практических и технических задач. Им свойственен дедуктивный способ мышления. В обучении они предпочитают практические эксперименты и лабораторные задания.

4.2.Модель Хани-Мамфорда

Развивая идеи Дэвида Колба, в 1986 году британские ученые Питер Хани (Peter Honey) и Алан Мамфорд (Alan Mumford) модифицировали эмпирическую модель обучения Колба. Стадии цикла обучения были переименованы в соответствии с управленческим опытом решения проблем и принятия решений: получение опыта (Experiencing), рассмотрение опыта (Reviewing), получение выводов из опыта (теоретизирование) (Theorizing) и планирование следующих шагов (Planning). Как правило, в общем цикле эмпирического обучения люди начинают обучение с предпочитаемого ими стиля. Стили обучения были напрямую связаны с вышеперечисленными стадиями и названы следующим образом: активист (Activist), мыслитель (Reflector), теоретик (Theorist) и прагматик (Pragmatist) [26].

Каждому из них присущи свои сильные и слабые стороны, свои особенности поведения, требования к процессу обучения и к другим его участникам. Учащиеся, предпочитающие тот или иной стиль в «чистом» виде, встречаются достаточно редко; как правило, у каждого в большей или меньшей степени представлены элементы всех стилей. Но именно доминирующие тенденции определяют и особенности процесса обучения и реакцию обучаемого на определенные методы преподавания [1].

Рис.5. Модель Хани-Мамфорда

Активисты предпочитают получать знания на основании своего опыта методом проб и ошибок. Они имеют широкие взгляды, с энтузиазмом относятся ко всему новому. Для них характерна тенденция сначала – действовать, а потом уже думать о

последствиях. Активисты общительны, активны, непредсказуемы, не склонны к скептицизму.

Активисты лучше обучаются, когда существуют новые ситуации, проблемы, задачи, на которых можно учиться. Происходят быстрые изменения, и приходится иметь дело с разнообразными задачами. Можно свободно генерировать идеи без необходимости подчиняться правилам. Когда в процессе обучения необходимо умение проявлять инициативу и активно действовать.

Мыслители предпочитают рассматривать проблему и ситуацию с разных точек зрения, «отстраненно», тщательно продумывая и оттягивая, насколько возможно, окончательный вывод. Прежде чем принять решение, предпочитают рассмотреть все возможные аспекты проблемы и собрать как можно больше информации. Они предпочитают наблюдать, а не действовать. Мыслители осторожны, вдумчивы и осмотрительны, немногословны в дискуссиях. *Мыслители* лучше обучаются, когда существует возможность для тщательного наблюдения и обдумывания; когда есть достаточно времени, чтобы подумать, прежде чем действовать, подготовиться, прежде чем выступать; когда в процессе обучения требуется провести тщательное исследование, т.е. собрать информацию, исследовать проблему; когда можно принимать решение в собственном ритме, без давления извне и жестких окончательных сроков.

Теоретики предпочитают получать знания структурированно, читать и глубоко разбираться в теории. Они систематизируют свои наблюдения, объединяют разрозненные факты в стройную теорию; решают проблемы на основе формальной логики, шаг за шагом, последовательно и планомерно; любят анализировать, склонны к системному мышлению, ценят рациональность и логику; стремятся к максимально возможной определенности, к совершенству и порядку во всем.

Теоретики лучше обучаются, когда обучение структурировано и имеет четко сформулированные идеи. Основной акцент в процессе обучения делается на рациональность и логику. Объект обучения дается в контексте систем, моделей, теории. Им необходимо достаточно времени для методичного исследования связей между идеями, событиями, ситуациями.

Прагматики в основном обучаются в процессе деятельности и основное внимание уделяют практике, а не теории. Они любят экспериментировать, стремятся проверить новые идеи на практике, быстро и уверенно работают над идеями, которые их привлекли. Прагматики рассматривают проблемы и возможности, как брошенный им

вызов, и верят в существование оптимального способа действий. Они хотят заниматься делом и не терпят долгих дискуссий и размышлений.

Прагматики лучше обучаются, когда существует очевидная связь между изучаемым предметом и проблемами, решаемыми в жизни: когда в процессе обучения демонстрируются техники, позволяющие получить конкретные практические результаты, когда можно опробовать новые методы на практике под руководством опытного наставника, когда существует возможность немедленного практического применения изучаемого.

4.3.Модель Фельдера-Сильверман

Самая известная классификация стилей обучения – это *модель Фельдера-Сильверман (Felder-Silverman model)*, предложенная американскими ученым Ричардом Фельдером (Richard Felder) и психологом Линдой Сильверман (Linda Silverman) в 1988 году [24]. Она строится на основе способов получения и обработки информации учащимися и основана на четырех факторах с двумя противоположными значениями:

- способ восприятия информации: сенсорный/интуитивный (sensing/intuitive);
- способ представления информации: визуальный/вербальный (visual/verbal);
- способ обработки информации: активный/рефлексивный (active/reflective);
- способ организации информации: последовательный/целостный (sequential/global).

Рис. 6. Модель Фельдера-Сильверман

Сенсорный и интуитивный способ восприятия информации. Учащиеся с сенсорным способом восприятия информации обладают конкретным мышлением, хорошо умеют работать с фактами и деталями, проводить эксперименты, используя при этом уже известные проверенные методы. Учащиеся с интуитивным способом восприятия, напротив, обладают абстрактным мышлением, предпочитают иметь дело с понятиями, идеями, теориями, отдают предпочтение инновационным подходам в обучении.

Визуальный и вербальный способ представления информации. Для учащихся, предпочитающих графики, диаграммы, рисунки, видео, анимации и вебинары, требуется визуальное представление информации. Если, к примеру, им объяснить правило, не подкрепив его графической схемой или рисунком, они могут не усвоить его. Напротив, учащиеся, отдающие предпочтение вербальному способу представления информации в устном и письменном виде, читая, проговаривая и записывая. Они предпочитают устные и письменные объяснения, иначе говоря, словесные описания, а не изображения.

Активный и рефлексивный способ обработки информации. Учащиеся, которым свойственен рефлексивный способ обработки информации, лучше воспринимают новый учебный материал в спокойной обстановке, предпочитают работать в одиночку, обдумывая при этом каждый свой шаг. Активные учащиеся, напротив, усваивают материал во время споров и дискуссий, в группе из нескольких человек, поэтому в дистанционном обучении для них важно наличие чатов и вебинаров. Они предпочитают сначала делать, а лишь потом оценивать результат.

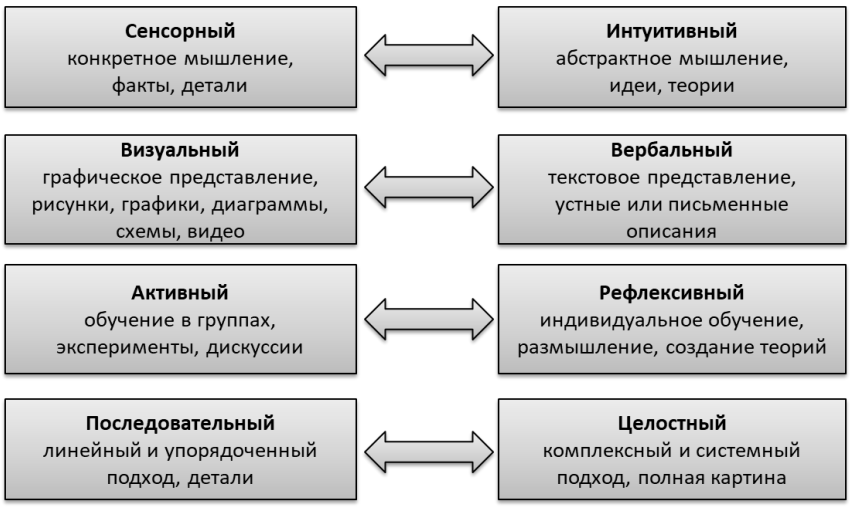


Рис. 7. Характеристики стилей обучения

Последовательный и целостный способ организации информации. Учащиеся, усваивающие информацию постепенно, пошагово, используя логику и линейные рассуждения, отдают предпочтение последовательному способу организации информации при обучении. Представление учебного материала для них должно быть непрерывным. Напротив, учащиеся, склонные к целостному восприятию информации, предпочитают системный подход, они видят полную картину, интегрируя и синтезируя отдельные знания. Они обучаются более стремительными темпами, сложные задачи любят решать нестандартными методами. Они могут овладеть тонкостями предмета,

только если понимают, как представляемый материал соотносится с тем, что они уже знали и изучали ранее, то есть пока они не охватят полную картину изучаемой темы [1].

4.4. Модель когнитивных характеристик

Другой подход, применяемый в адаптивных системах обучения – это адаптация к когнитивным способностям учащегося, как принято в *модели когнитивных характеристик (Cognitive Trait Model, CTM)* [31].

Это четыре когнитивных способности: объем рабочей памяти, способность к индуктивному рассуждению, способность к ассоциативному обучению и скорость обработки информации. Эти когнитивные способности, характеризующие общую познавательную способность учащегося, называются *когнитивными характеристиками*. Адаптация в этом контексте гарантирует, что познавательные способности учащегося не будут перегружены [1].

Объем рабочей памяти – определяющая способность «держать в уме» небольшие фрагменты информации, необходимые для сиюминутной мыслительной деятельности, например, для решения логической задачи или осознания сложной информации. *Способность к индуктивному рассуждению* – способность получать общие заключения из конкретных, или частных, суждений. *Способность к ассоциативному обучению* – способность устанавливать связи между новыми и уже известными понятиями. *Скорость обработки информации* – это когнитивная способность, которая может быть определена как время, необходимое человеку для решения умственной задачи. Этот навык связан со скоростью, с которой человек улавливает информацию и реагирует на неё, как с помощью зрения (через буквы и цифры), так и с помощью слуха (речь) или движения. Т.е. скорость обработки информации – это время между получением стимула и ответом на него.

Одна из интеллектуальных систем обучения, в которой применяется модель когнитивных характеристик – Marginal Costing Adaptive Learning Modules (MCALM) [33].

4.5. Таксономия Блума

Вопросы выявления, измерения и оценки уровня сформированности у учащихся знаний, умений и навыков в настоящее время являются одними из центральных в практике обучения. Если цель обучения определяет, что должен знать, уметь обучаемый, то задачи обучения отвечают на вопрос, как двигаться к цели.

В рамках образовательной технологии американским психологом методов обучения Бенджамином Блумом (Benjamin Samuel Bloom) в 1956 г. была создана первая

таксономия целей обучения [16]. При этом Б. Блум и Д. Кратволь разделили цели обучения на три области: когнитивную (требования к освоению содержания предмета), психомоторную (развитие двигательной, нервно-мышечной деятельности) и аффективную (эмоционально-ценностная область, отношение к изучаемому).

Таксономия Блума предоставляет четкую и стабильную основу для разработки адаптивных обучающих систем. Она является самой известной моделью, описывающей процесс мышления, и включает шесть навыков мышления, структурированных от самого базового до самого продвинутого уровня.

В своем фундаментальном труде «Таксономия учебных целей: сфера познания» Блум попытался сконструировать иерархию учебных целей, охватывающих когнитивную область, которая шаг за шагом описывала бы уровни человеческого мышления и вытекающие отсюда задачи обучения. С точки зрения Блума, цели обучения напрямую зависят от иерархии мыслительных процессов, их еще называют элементами таксономии Блума: *знание (knowledge), понимание (comprehension), применение (application), анализ (analysis), синтез (synthesis)* и *оценка (evaluation)* [1,16].

Рис. 8. Таксономия Блума

Этот список мыслительных процессов иерархически организован, начиная с самого простого, припоминания знания, до наиболее комплексного, состоящего в выработке суждений о ценности и значимости той или иной идеи.

Большинство мыслительных процессов, характерных для учебной деятельности в традиционной школе, соответствуют уровням знания или понимания, они являются наиболее распространенными из мыслительных умений. Они служат базой или фундаментом, на котором строятся все мыслительные умения более высокого порядка. С каждым последующим уровнем мыслительные умения становятся более сложными и используются реже.

На первом, самом низшем уровне, *уровне знания (Knowledge Level),* деятельность учащегося сводится лишь к запоминанию и воспроизведению изучаемого материала. Речь идет о запоминании и воспроизведении конкретных фактов, методов и процедур, основных понятий, правил, принципов, целостных теорий. Соответственно каждому уровню с помощью определенных глаголов может предлагаться набор задач. Так, например, для уровня знания подойдут задачи, начинающиеся с глаголов *запомните, повторите, перечислите, назовите, напишите, определите, выучите* и т.д.

Второй уровень, *уровень понимания (Comprehension Level),* заключается в умении понимать значение, перефразировать главную мысль. Понимание достигается путем объяснения, описания, определения, обсуждения, формулирования, иллюстрирования, демонстрации. Показателем понимания смысла изученного является способность устанавливать связь одного материала с другим, превращать его из одной формы выражения в другую, переводить из одной формы в другую (например, из текстовой в графическую, математическую и наоборот). Показателем понимания может также быть интерпретация материала учащимся (объяснение, краткое изложение), прогнозирование последствий, вытекающих из имеющихся данных.

Третий уровень, *уровень применения полученных знаний (Application Level),* требует от учащегося умений использовать изученный материал в конкретных условиях и новых ситуациях. Сюда входит применение правил и методов, умение разбивать материал на составляющие понятий, законов, принципов, теорий. Задачи, нацеленные на применение знаний, формулируются с помощью глаголов *решать, планировать, объяснять, изображать, экспериментировать, тренироваться, показывать, использовать, учить, демонстрировать*.

Уровень анализа информации (Analysis Level) заключается в умении разбить материал на составляющие и выявить взаимосвязи между ними и принципы построения так, чтобы выявить структуру материала. Учебные результаты характеризуются осмыслением не только содержания учебного материала, но и его внутренней структуры. Учащийся, который хорошо овладел этой категорией учебных целей, видит ошибки в логике рассуждений, разницу между фактами и последствиями, оценивает

значимость данных. Аналитические способности формируются заданиями с ключевыми глаголами *исследовать, сравнивать, противопоставлять, разделять, интерпретировать, анализировать, группировать, отбирать, классифицировать* и т.д.

Уровень синтеза (Synthesis Level) направлен на формирование навыков обобщения, соединения идей для создания чего-то нового. Этот уровень состоит в умении комбинировать элементы, чтобы получить целое с новым системный свойством. Таким новым продуктом может быть разработка плана и возможной системы действий, получение системы абстрактных отношений и т.д. Способности к синтезу тренируются задачами, ориентированными на составление, сочинение, соединение, конструирование, воображение, формулирование, построение, изобретение.

Уровень оценки полученных знаний (Evaluation Level) призван формировать навыки мышления, которые помогают учащемуся делать суждения относительно ценности полученной информации. Этот уровень направлен на самостоятельную интеллектуальную деятельность и требует умения делать заключения на основе имеющихся данных и внешних критериев, оценивать, одобрять, поддерживать, рекомендовать, критиковать и делать выводы. Как категория учебных целей, она означает умение оценивать значение того или иного материала для конкретной цели. Суждения и умозаключения учащегося должны основываться на четких критериях. Учащийся оценивает логику построения материала в виде письменного текста, оценивает соответствие выводов данным и т.д.

По Блуму, каждый уровень этой когнитивной пирамиды базируется на предыдущем. В основе всего лежит знание, а наивысшей точкой как когнитивных способностей, так и целей обучения является способность к независимой оценке. Т.е. без знания невозможно понимание, без понимания невозможно использование, без освоения начальных уровней невозможен анализ и синтез, а без всего этого невозможна творческая оценка явлений и событий.

5. Заключение

В статье рассмотрены вопросы реализации адаптивности в интеллектуальных системах дистанционного обучения, основанной на технологии адаптивной гипермедиа и моделях индивидуальных стилей обучения. Разработка адаптивных обучающих гипермедиа-систем, в которых процесс обучения и проверки знаний адаптируется, основываясь на индивидуальных характеристиках учащихся, является актуальным и перспективным направлением современных исследований.

В рамках статьи представлены различные методы и техники адаптивного представления и адаптивной навигации, приведены основные модели, учитывающие индивидуальные стили обучения. Теория стилей обучения основана на том, что разным людям подходят различные стили обучения, и процесс обучения будет более эффективным, если им будут даны подходящие инструкции, адаптированные к их стилю обучения. Гипермедиа пространство адаптивных гипермедиа-систем адаптируется в соответствии с моделью учащихся, предоставляя адаптивное содержание и адаптивную навигацию, основываясь на индивидуальных стилях обучения, под которыми понимаются различные способы получения, обработки и хранения информации учащимися.

Список литературы

1. Волянская Т.А. Адаптивное генерирование учебных курсов в интеллектуальных системах дистанционного обучения. Часть 1. – Новосибирск, 2019. – 54 с. – (Препр. / ИСИ СО РАН; № 183).
2. Волянская Т.А. Адаптивное генерирование учебных курсов в интеллектуальных системах дистанционного обучения. Часть 2. – Новосибирск, 2019. – 40 с. – (Препр. / ИСИ СО РАН; № 184).
3. Волянская Т.А. Виртуальный музей истории информатики в Сибири // Материалы Междунар. конф. молодых ученых по математическому моделированию и информационным технологиям. – Новосибирск, 2002. – С. 49.
4. Волянская Т.А. Виртуальный музей истории информатики в Сибири: модель предметной области и модель пользователя // Новые информационные технологии в науке и образовании. – Новосибирск, 2003. – С. 124–146.
5. Волянская Т.А. Методы адаптации гипермедиа и их применение при создании виртуального музея истории информатики в Сибири // Материалы XL Междунар. науч. студенческой конф. «Студент и научно-технический прогресс». – Новосибирск, 2002. – С. 173–174.
6. Волянская Т.А. Методы и технологии адаптивной гипермедиа // Современные проблемы конструирования программ. – Новосибирск, 2002. – С. 38-68.
7. Волянская Т.А., Касьянов В.Н., Несговорова Г.П. Адаптивная гипермедиа и ее использование при создании виртуального музея истории информатики в Сибири // Материалы Пятой международной конференции «Перспективы систем информатики» PSI’03. – Новосибирск, 2003. – С. 10-12.

8. Касьянов В.Н. Музеи и Интернет: новая виртуальная реальность // Вычислительные технологии, 2008, Том 13, С. 239-247.
9. Касьянов В. Н, Касьянова Е. В. Адаптивные системы и методы дистанционного обучения // Информационные технологии в высшем образовании. – 2004. – Т.1, № 4. – С. 40–60.
10. Касьянов В. Н., Несговорова Г. П., Волянская Т. А. Виртуальный музей истории информатики в Сибири // Проблемы программирования, 2003, N 4, С. 82-91.
11. Касьянова Е.В. Адаптивные методы и средства поддержки дистанционного обучения программированию. – Новосибирск: ИСИ СО РАН, 2007. – 170 с.
12. Касьянова Е.В. Методы и средства обучения программированию в вузе // Образовательные ресурсы и технологии. – 2016. – № 2. – С. 23–30.С. 10–12.
13. Anderson J.R. The expert module // Foundations of Intelligent Tutoring Systems / Eds. Polson M.C., Richardson J. J. – Lawrence Erlbaum Associates Publishers, London, 1988. – P. 21–53.
14. Beaumont I., Brusilovsky P. Educational applications of adaptive hypermedia // Human-Computer Interaction, Proceedings of Interact’95, Lillehamer, Norway. – London, Chapman & Hall. – 1995. – P. 410–414.
15. Benyon D., Murray D. Adaptive systems: From intelligent tutoring to autonomous agents // Knowledge-Based Systems. – 1993. – Vol. 6, N 4. – P. 197–219.
16. Bloom B.S. Taxonomy of educational objectives: The classification of educational goals: Handbook I, cognitive domain. – New York, Toronto: Longmans, Green, 1956. – 207 p.
17. Brusilovsky P. Adaptive educational systems on the world-wide-web: A review of available technologies // Proceedings of Workshop "WWW-Based Tutoring" at 4th International Conference on Intelligent Tutoring Systems (ITS'98). – San Antonio, 1998.
18. Brusilovsky P. Methods and Techniques of Adaptive Hypermedia // User Modeling and User Adapted Interaction. – 1996. – Vol. 6, N 2–3. – P. 87–129.
19. Brusilovsky P., Nijhawan H. A framework for adaptive e-learning based on distributed re-usable learning activities // Proceedings of World Conference on E-Learning, E-Learn 2002. – Montreal, Canada, AACE, 2002. – P. 154–161.
20. Brusilovsky P., Schwarz E., Weber G. ELM-ART: An intelligent tutoring system on World Wide Web // Intelligent Tutoring Systems, Springer, 1996. – P. 261–269.

21. Burns H.L., Capps C.G. Foundations of intelligent tutoring systems: An introduction // Foundations of Intelligent Tutoring Systems / Eds. Polson M. C., Richardson J. J. – Lawrence Erlbaum Associates Publishers, London, 1988. – P. 1–19.
22. Corno L., Snow R.E. Adapting teaching to individual differences among learners // Handbook of research on teaching, 3. – 1986. – P. 605–629.
23. Cronbach L., Snow R. Aptitudes and Instructional Methods: A Handbook for Research on Interactions. – New York, NY, USA: Irvington, 1977. – 592 p.
24. Dagger D., Wade V., Conlan O. Developing Adaptive Pedagogy with the Adaptive Course Construction Toolkit (ACCT) // Proceedings of the Third International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, AH2004 / Eds. De Bra P. and Nejdl W. – Berlin: Springer Verlag, Eindhoven, The Netherlands, 2004. – P. 55–64.
25. Felder R.M., Silverman L.K. Learning and teaching styles in engineering education // Engineering education. – 1988. – Vol. 78, N 7. – P. 674–681.
26. Honey P., Mumford A. The manual of learning styles. – Maidenhead, Peter Honey, 1992.
27. Kasyanov V.N. An open adaptive virtual museum of informatics history in Siberia // IFIP International Federation for Information Processing. - Boston: Springer, 2008. - Vol. 266. History of Computing and Education 3 (HCE 3). - p. 129-146. - (Proc. of the 20th IFIP World Computer Congress).
28. Kasyanov V. SVM - Siberian Virtual Museum of Informatics History // Innovation and the Knowledge Economy: Issues, Applications, Case Studies, Amsterdam, IOS Press, 2005, Part 2, p.1014-1021.
29. Kasyanov V.N., Kasyanova E.V. A Web-based system for distance learning of programming // Lecture Notes in Electrical Engineering. - Springer, 2009. - Vol. 27. - pp. 453-462. - (Proceedings of the European Computing Conference).
30. Kasyanov V., Kasyanova E. WAPE — a system for distance learning of programming // Learning to Live in the Knowledge Society: IFIP 20th World Computer Congress. — Boston: Springer, 2008. – P. 355–356. – (IFIP International Federation for Information Processing, Vol. 261).
31. Kinshuk, Lin T., Patel A. Supporting the Mobility and the Adaptivity of Knowledge Objects by Cognitive Trait Model // Innovations in instructional technology: Essays in honor of M. David Merrill / Eds. Spector J. M., Ohrazda C., Van Schaack A., and Wiley D. – Mahwah, NJ: Lawrence Erlbaum, – 2005. – P. 29–41.

32. Kolb D. Experiential Learning: Experience as the source of Learning and Development. – Englewood Cliffs, New Jersey: Prentice-Hall, 1984.
33. Lin T. Cognitive Trait Model for Adaptive Learning Environments: PhD Thesis, Massey University. – Palmerston North, New Zealand, 2007.
34. Mödritscher F. Adaptive E-Learning Environments: Theory, Practice, and Experience. – Verlag Dr. Müller, 2008.
35. Oppermann R., Rasher R. Adaptability and adaptivity in learning systems // Knowledge Transfer. – 1997. – Vol 2. – P. 173–179.
36. Rickel J.W. Intelligent computer-aided instruction: A survey organized around system components // Systems, Man and Cybernetics, IEEE Transactions on Systems, Man, and Cybernetics. – 1989. – Vol. 19, N 1. – P. 40–57.
37. Sleeman D., Brown J.S. Introduction: Intelligent Tutoring Systems: An Overview // Intelligent Tutoring Systems. – Academic Press, Burlington, MA, 1982. – P. 1–11.
38. Snow R.E., Swanson J. Instructional psychology: Aptitude, adaptation, and assessment // Annual Review of Psychology. – 1992. – Vol. 43, N 1. – P. 583–626.
39. Stankov S., Grubišić A., Žitko B. E-learning paradigm & Intelligent tutoring systems // Annual 2004 of the Croatian Academy of Engineering. – 2004. – P. 21–31.
40. Shute V.J., Lajoie S.P., Gluck K.A. Individualized and group approaches to training // Training and retraining: A handbook for business, industry, government, and the military. – 2000. – P. 171–207.
41. Shute V.J., Psotka J. Intelligent Tutoring Systems: Past, Present and Future // Eds. Jonassen D. / Handbook of Research on Educational Communications and Technology. – New York, NY: Macmillan, 1996. – P. 570–600.
42. Shute V., Towle B. Adaptive e-learning // Educational Psychologist. – 2003. – Vol. 38, N 2. – P. 105–114.
43. Uhr L. Teaching machine programs that generate problems as a function of interaction with students // Proceedings of the 24th ACM National Conference. – 1969. – P. 125–134.
44. Wenger E. Artificial Intelligence and Tutoring Systems. – Morgan Kaufmann Publishers, Inc., California, USA, 1987.
45. Woolf B. AI in Education // Encyclopedia of Artificial Intelligence / Ed. Shapiro S. – John Wiley & Sons, Inc., New York, 1992. – P. 434-444.

УДК 004.67

Некоторые эксперименты по построению и анализу графа Де Брёйна

Марчук А.Г. (Институт систем информатики СО РАН)

Трошков С.Н. (Институт систем информатики СО РАН)

В статье описывается опыт решения задачи нахождения цепочек в графе Де Брёйна с применением параллельных вычислений и распределенным хранением данных.

Ключевые слова: граф, распределенные системы, dotnet, Spark, Scala

1. Введение

Одной из важных задач в современной биоинформатике является сборка генома по многочисленным прочтениям фрагментов ДНК. Задача, о которой пойдет речь в статье – начальный этап задачи о сборке генома. Для простоты, можно представить геном как длинную строку символов, где каждый символ соответствует азотистому основанию, из которого состоит ДНК. Аппаратура, которую используют биологи, позволяет считать множество маленьких частей этой строки, и наша цель – восстановить полную строку по ним.

Существует множество программ для сборки генома, но эффективного распределенного решения для этой задачи нет. Между тем в нём есть потребность, так как объем данных, с которыми сталкиваются биологи, может превосходить физически возможный объем данных для одного компьютера.

Авторы выражают благодарность Юрию Вяткину за помощь в постановке задачи и Денису Мигинскому за участие в обсуждении способов решения и полученных результатов.

2. Задача

Мы будем решать часть задачи сборки генома, а именно нахождение контигов [3]. Контиг представляет собой набор перекрывающихся сегментов ДНК, которые в совокупности представляют собой консенсусную область ДНК. Опуская биологические подробности, сформулируем задачу о нахождении контигов, используя математические объекты и термины.

На вход подаются строки разной длины из символов алфавита A, C, G, T.Далее выбирается некоторое число K, меньшее, чем длина самой короткой из строк. Каждая из данных строк разбивается на подстроки длины K. Эти подстроки - вершины нашего графа, он называется граф Де Брёйна. Между двумя вершинами проводится грань, если в исходной строке эти подстроки пересекаются по K-1 символам. Нужно найти все цепочки вершин без разветвлений и свернуть их в одну вершину. Получившиеся вершины называются контигами. Таким образом, по заданному набору данных секвенирования, требуется построить граф Де Брёйна и найти некоторое число наиболее длинных контигов.

3. Решение Apache Spark

Для первой группы экспериментов, в качестве платформы для разработки был выбран Apache Spark [6]. Apache Spark позволяет создавать приложения для кластерных систем, не задумываясь о многопоточности и распределении процессорных ресурсов, при условии что вы используете специальные структуры данных Spark для хранения и обработки данных.

3.1. Структуры данных Apache Spark

RDD(Resilient Distributive Dataset) Впервые представлен в версии Apache Spark 1.0. Распределенная коллекция данных, расположенных по нескольким узлам кластера, набор объектов Java или Scala, представляющих данные. RDD [5] работает со структурированными и с неструктурированными данными.

DataFrame Впервые представлен в версии Apache Spark 1.3. Распределенная структура данных, во многом похожая на таблицу в реляционной БД. Поддерживает язык запросов SQL [1].

Для решения задачи была выбрана именно DataFrame, как более современная структура с богатым набором функций и оптимизаций. В частности, при тестировании выяснилось, что DataFrame занимает примерно в два раза меньше объема оперативной памяти, чем RDD с такими же данными. К тому же, любой RDD можно перевести в DataFrame, а любой DataFrame можно перевести в RDD, хоть эта операция и может быть затратной на больших данных. Единственной проблемой при этом было то, что встроенная библиотека для работы с графами GraphX работает именно на RDD. Но существует аналог такой библиотеки для DataFrame – библиотека GraphFrames, которая в будущем, возможно, станет

частью Apache Spark.

3.2. Операции в Apache Spark

Операции в Apache Spark делятся на трансформации и действия.

Трансформация – это массовое преобразование, применяемое ко всей структуре данных. Например, фильтрация результатов по какому-либо критерию, агрегация, слияние двух структур данных, операция map. При вызове трансформации Spark запоминает ваш вызов, но вычислений не производит.

Реальные вычисления же начнутся только с вызовом действия на структуре данных. При этом Spark попытается оптимизировать порядок выполнения трансформаций, без ущерба выходным данным. К действиям относятся подсчет полей, вывод, операция reduce.

Внутри трансформаций вызывать другие трансформации нельзя. Важно, что операция cache(), которая помещает структуру данных в кэш оперативной памяти, считается трансформацией, а не действием. Это значит, что данные будут закэшированы не во время вызова операции, а лишь после вызова следующего действия.

3.3. Работа с Apache Spark

Apache Spark можно запускать на системах управления кластерами Apache Mesos, Kubernetes, YARN, либо в режиме Standalone без систем управления кластерами. Режим Standalone является самым простым для настройки, поэтому он был выбран для проведения экспериментов. Для тестирования использовались 4 виртуальные машины, для каждой машины было отведено 4 ядра процессора Inter Xeon CPU E5-2680 2.7 GHz и 20 ГБ оперативной памяти.

Для решения задачи был составлен итерационный алгоритм. На каждой итерации проверялось является ли очередная вершина началом цепочки. Если она является таковой, то происходит обход цепочки, удаление лишних вершин и переименование исходной вершины. В результате, после обхода всех вершин, получаем граф без цепочек. Схема работы алгоритма представлена на Рис. 1.

Итерировать ряды DataFrame по одному можно с помощью функции map или foreach. Эти функции являются трансформациями, а потому итерировать ряды возможно, но использовать для преобразований информацию из этого же или других DataFrame весьма проблематично. Поэтому в первой версии написанной программы итерирование произ-

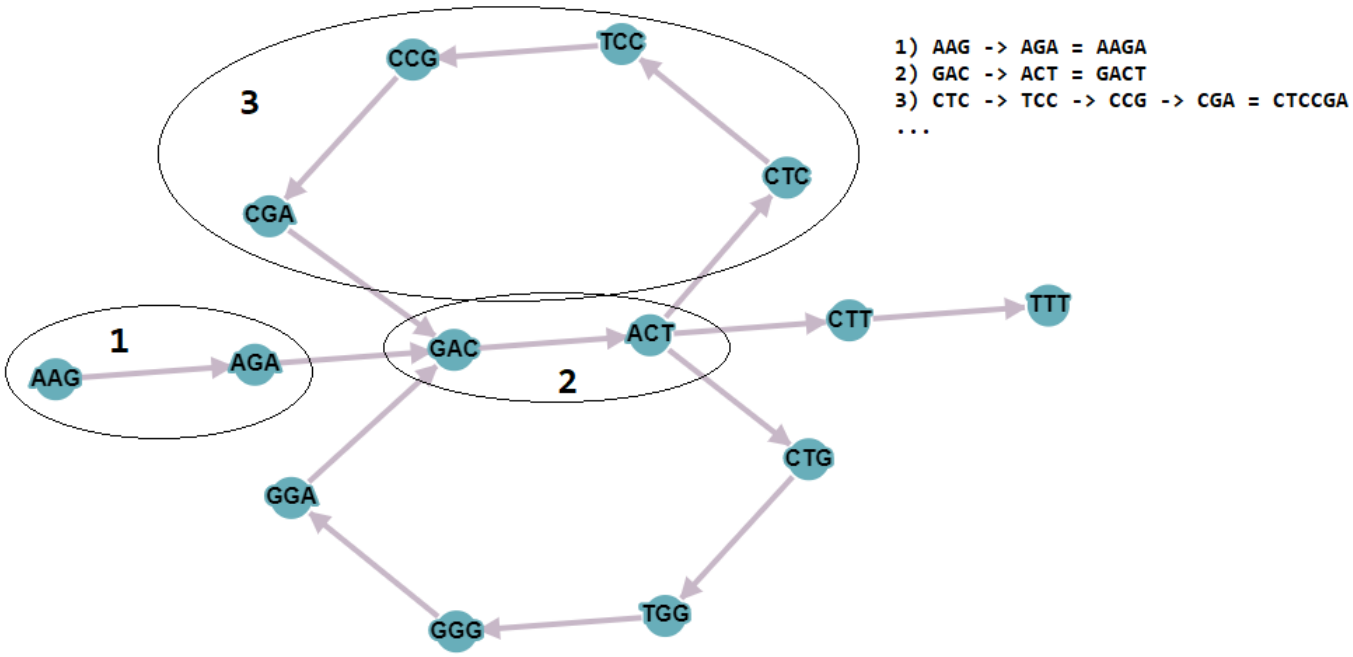


Рис. 1. Первая версия алгоритма

водилось внешним циклом языка Scala, в котором преобразовывался очередной ряд из DataFrame, в зависимости от ссылок в нем. По нагрузке процессоров было выявлено, что такая программа работает на одном компьютере, и лишь обращается к другим за данными. Это было логично, ведь задачи в Spark будут выполняться параллельно, только если это трансформации и действия над большим объемом данных. Требовалось изменить алгоритм, чтобы он работал сразу с несколькими записями, а не с каждой по очереди.

Далее описывается принцип работы новой версии алгоритма. Для начала находятся все ребра, являющиеся частью цепочки в графе. То есть, ребра, в которых у начальной вершины исходящая степень равна 1, а у конечной вершины входящая степень равна 1. После, среди найденных ребер нужно найти ребра, которые являются началом цепочки. Это будут такие ребра, исходящих вершин которых нет среди множества входящих вершин ребер, найденных нами на предыдущем шаге алгоритма. Следующим шагом нужно «склеить» вершины начальных ребер, то есть совместить их исходящую и входящую вершину в одну. Проделав эти шаги, мы укоротили все цепочки в графе на одно ребро несколькими массовыми операциями. Эти действия нужно повторять рекурсивно, пока не иссякнет множество ребер, являющихся цепочками в графе. На выходе множество вершин и будет являться искомыми контигами. Схема работы алгоритма представлена на Рис. 2.

4. Решение C#

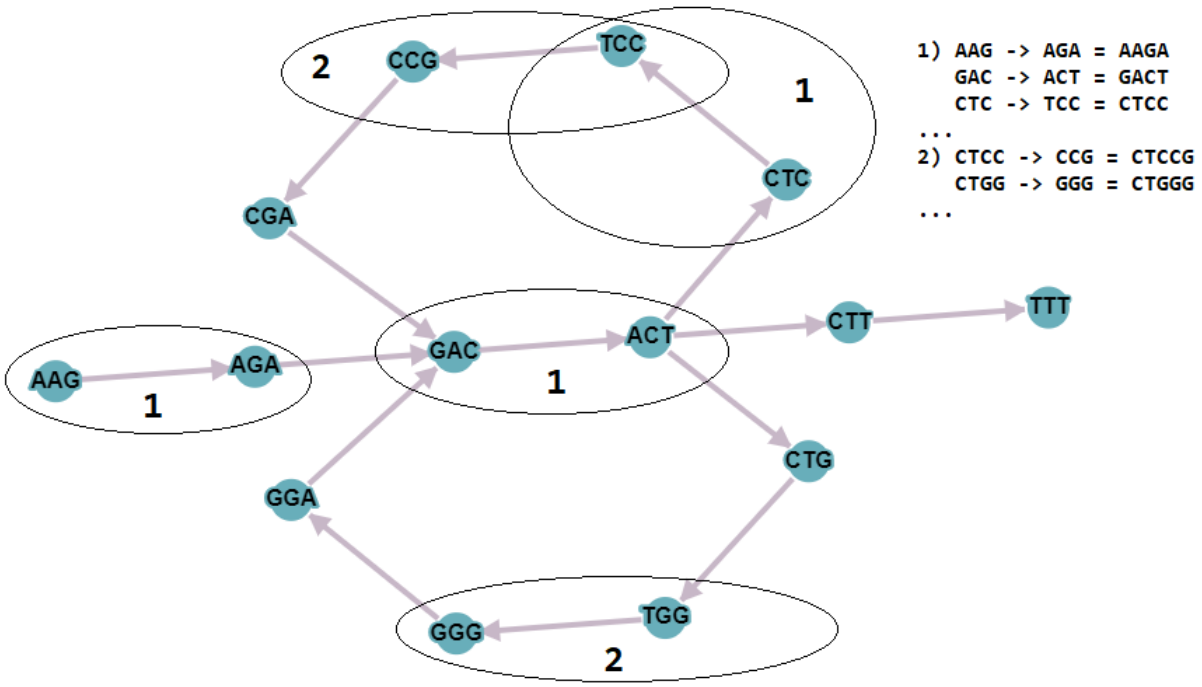


Рис. 2. Вторая версия алгоритма

Время работы программы, написанной в среде SPARK, показалось огромным для такой относительно простой задачи. Для второй группы экспериментов было принято решение написать программу на языке C#, которая решала бы эту же задачу как можно проще, не используя к сложные структуры данных, такие как Dataframe.

Для проверки предельных возможностей работы с графом Де Брёйна, были созданы две программы DeBruijnOrtho и DeBruijnDirect. Программы создавались с использованием библиотеки PolarDB [4] и с некоторым упрощением графа. Упрощение заключалось в том, что в графе не строились дуги между узлами если узел-источник дуги имеет больше одного наследника или узел-приемник дуги имеет больше одного предшественника. Полученный граф представляет собой множество линейных цепочек узлов, связанных отношением соседства. Нас интересовали стадии создания графа и вычисления максимальной (самой длинной) цепочки. Если задачу обработки графа Де Брёйна ограничить только вычислением контига максимальной длины или даже всех контигов, то указанное упрощение графа эквивалентно исходному.

Программы создавались «вручную» достаточно низкоуровневыми структурами и средствами обработки (файлы, массивы, структуры, стандартная библиотека, библиотека PolarDB). Программа DeBruijnOrtho делалась для работы в сетевой среде с использованием средств бинарного обмена информацией TCP. Для оценки пределов производи-

тельности для однозадачной (однопроцессорной) конфигурации, была сделана программа DeBruijnDirect. Существенная часть ее в какой-то мере повторяет решения DeBruijnOrtho, но полностью исключена сетевая часть и произведены специфические оптимизации.

4.1. Сетевое устройство программ

Рассмотрим некоторые особенности программ и некоторые примененные способы оптимизации. Отметим, что программа ориентирована на обработку традиционных семейств данных (ридов), возникающих в современных процессах секвенирования геномов живых организмов. Причем оптимизация велась как в сторону уменьшения используемых ресурсов, в основном памяти, так и в сторону ускорения вычислений. Предполагалось, что обработку данных уровня человеческого генома (оценочно, это 3.5 млрд. узлов графа) можно будет выполнять на одном компьютере со средним объемом ОЗУ или на относительно небольшом высокопроизводительном кластере. Большой (иногда огромный) объем начальных и промежуточных данных составляет главную проблему.

Программа организована как множество задач (процессов) обработки, расположенных на разных компьютерах. Одна из запускаемых задач называется «мастером», остальные называются исполнителями. Технически, запускается один и тот же код (программа), но на соответствующих компьютерах и с соответствующими параметрами. По параметрам, программа определяет запущена ли она как мастер или как исполнитель.

Сначала запускается мастер, ему никого ждать не надо, он сам ждет, когда с ним свяжутся. Исполнителю дается IP-адрес мастера и он устанавливает связь с ним. Связь между мастером и исполнителем работает отдельными командами, посылаемыми мастером. То есть, мастер посылает 1 байт – это команда , потом он посылает фактические значения аргументов команды и ждет результатов. Принимает результаты и на этом цикл команды заканчивается и исполнитель ждет следующую команду. Аргументов и результатов может не быть, но если они запланированы, то они обязаны появиться и в нужном формате.

Все общение производится в бинарном виде средствами BinaryWriter и BinaryReader. Посылаются любые примитивные типы (байт, целые разных размеров, строки и т.д.). Последовательность посылается посылкой двойного целого с числом элементов и потом подряд идут элементы последовательности. Запись - посылаются подряд идущими элементами записи. В общем - как в Поляре. В дальнейшем предполагается внедрить типовую систему Поляра, это позволит усилить контроль за коммуникациями.

Обработка данных выполняется на мастере. На мастере выполняются преобразования, требующие поэлементной потоковой обработки и на мастере организуются вычисления, требующие привлечения исполнителей. Исходными данными является набор ридов, задаваемый в текстовом виде во входном файле и расположенный на мастере.

4.2. Получение узлов графа

Первый этап - преобразование ридов в бинарную форму. Это потоковая обработка, когда последовательность строк ридов преобразуется в последовательность слов с заданным окном сканирования. Окно - это диапазон задаваемой длины n выборки последовательности символов. Для строки рида S первое слово будет S[0], ... S[n-1], второе - S[1], ... S[n] и так далее. Слова кодируются в двойные целые (UInt64) каждый символ - 2 разряда, от младших разрядов к старшим. Соответствие символа 2-битному коду следующее: A-0, C-1, G-2, T-3. Есть два варианта, ограничивающие размер окна в 32 и 64 символа. В дальнейшем, предполагается перейти к формату набора байтов. Таким образом, результатом первого этапа обработки является последовательность последовательностей кодированных слов. Полученный массив также располагается на мастере. В принципе, получаемый массив - довольно большой. Для набора из 500 тыс. ридов, занимающего 49 мб получается файл размером 319 мб. Для задачи в 1000 раз большей, это уже будет представлять проблему.

Следующий этап - перевод бинарных ридов в кодированные. Кодированные риды - это те же наборы слов, но закодированные целочисленным кодом. Кодирование - взаимнооднозначное, т.е. по слову однозначно определяется код, по коду - слово. Первое соответствие после второго этапа не сохраняется. Второе - сохраняется. Главное, что кодирование обеспечивает то, что 2 разных слова имеют разные коды и 2 одинаковых - одинаковые. Эффективное кодирование требует довольно много оперативной памяти, напр. для 500 тыс. ридов может потребоваться около 2 Гб. ОЗУ. Применены 2 приема для снижения нагрузки на память. Во-первых, кодирование выполняется в несколько проходов, во вторых - с привлечением исполнителей.

Основная операция, применяемая на этой стадии:

$IEnumerable < int > GetSetNodes(IEnumerable < UInt64 > bwords)$;

В ней поток бинарных слов преобразуется в поток их кодов с одновременным пополнением хеш-таблицы. На каждом из проходов преобразуется в коды лишь часть слов, соответственно объем хеш-таблицы уменьшен. Причем по завершению прохода, мы мо-

жем уничтожить таблицу имен потому что в оставшейся части таких слов не остается. Многопроходный подход "конвертирует"дополнительное время на обработку в уменьшение расхода оперативной памяти. Использование исполнителей основывается на том, что в каждом исполнителе накапливается набор слов, обладающих некоторыми свойствами типа word%n-1 == номер секции (исполнителя). Соответственно, в указанном методе GetSetNodes набор слов разбивается по секциям и выполняется обращение к этим секциям, а полученные результы сливаются. В результате данного этапа формируется последовательность (для кластерного варианта - множество последовательностей) узлов, в которой номер элемента последовательности является кодом слова. Соответственно, на этом этапе, слова превращаются в узлы.

Другая часть графа - ребра, реализуется (в упрощенном представлении) через ссылки (коды узлов) "вперед"и "назад в структуре узла CNode. Это делается через модификацию списка узлов при сканировании файла кодированных ридов. Следующий этап – нахождение начал непрерывных цепочек. Основная идея в том, что цепочка может начинаться только с узла у которого нет предыдущего или предыдущих несколько или предыдущий один, но у него несколько следующих. Кроме того, нет смысла рассматривать те узлы, у которых не единственный следующий. В нормальных данных число непрерывных цепочек (контигов) существенно меньше числа узлов, поэтому такой список оперативную память не нагружает.

Далее, идет этап отслеживания цепочек с выделением самой длинной. Собственно идет продвижение от начального узла по указателям на следующий до пропажи указателя. Если при этом, будет превышено количество пройденных узлов в данной цепочке, то лучшая цепочка заменяется на текущую. Главная проблема процесса прохождения цепочек – медленность чтения следующего указателя из последовательности узлов графа, поскольку очередной узел может располагаться в произвольном месте последовательности, а значит – файла последовательности. Ускорение этого процесса осуществляется запуском одновременного отслеживания большого числа цепочек с последующим группированием запросов на получение следующего узла. Что это дает? Если граф разбит на секции, а секции располагают узлы в более «близких» местах, напр. на одном вычислителе, то сгруппированные запросы снова разбиваются теперь уже по секциям и выполняются более эффективно и, возможно, в параллель.

5. Сравнение двух подходов

В таблицах ниже представлено общее время работы программ на кластере в зависимости от размера входных данных и количества виртуальных машин в кластере, работающих над задачей. В работе мы использовали как реальные входные данные, так и специально сгенерированные для тестирования. Конкретно, 500 тысяч ридов и 14 миллионов ридов взяты из реальных ДНК вируса свиного гриппа и бактерии E. Coli соотвественно. 5 и 10 миллионов ридов – данные, которые были получены искусственно, с помощью генератора.

Программа, написанная с помощью Apache Spark, не работала со сгенерированными данными и с меньшим количеством машин. В этих случаях сильно вырастала нагрузка на сеть, и временные файлы для передачи данных переполняли всю физическую память машин.

Решение C#	500 тыс. ридов	5 млн. ридов	10 млн. ридов	14 млн. ридов
1 Машина	44s	5m 10s	10m 42s	15m 4s
2 Машины	1m 32s	8m 54s	17m 3s	23m 46s
4 Машины	1m 44s	11m 40s	22m 32s	29m 29s
8 Машин	2m 26s	19m 57s	33m 52s	38m 30s
16 Машин	5m 16s	76m 31s	111m 16s	88m 45s

Решение Apache Spark	500 тыс. ридов	14 млн. ридов
8 Машин	3h 24m	5h 18m
16 Машин	2h 35m	3h 1m

Все тесты проводились на кластере, собранном из виртуальных машин. Каждая из виртуальных машин находилась физически на одном и том же жестком диске, и сеть между ними была виртуальная, поэтому в наших решениях мы опустили оценку скорости передачи данных по сети.

Несмотря на то, что решение Apache Spark работает в разы медленнее, именно на нём можно заметить рост производительности с увеличением количества машин, то есть выигрыш от распределенных вычислений. Время работы программы на C# увеличивается с добавлением машин, но всё же значительно превосходит Apache Spark по времени, и при работе на одной машине, оно даже сравнимо с известным решением SPAdes [2]. В нашем тесте на 500 тысяч ридов программа SPAdes нашла контиги за 15 минут, 13 из которых занимал обязательный этап исправления ошибок в ридах. Так как в нашей программе не учитываются разные биологические эвристики и нет этапа исправления ошибок, можно

предположить, что наше решение будет не хуже.

Список литературы

1. Amburst M., Xin R. Spark SQL: Relational Data Processing in Spark // SIGMOD '15: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. May, 2015. P. 1383–1394.
2. Bankevich A., Nurk S. SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing // Journal of computational biology : a journal of computational molecular cell biology, №19. 2012. P. 455-477.
3. Gregory S. Contig Assembly // Encyclopedia of Life Sciences. -2005.
4. Marchuk A.G. PolarDB: an infrastructure for specialized NoSQL databases and DBMS // Automatic Control and Computer Sciences. - 2016. - V. 50. - № 7. - P. 493-496.
5. Matei Zaharia, Mosharaf Chowdhury. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing // University of California, Berkeley. Technical Report No. UCB/EECS-2011-82. -2011.
6. Srinivas Jonnalagadda V., Srikanth P. A Review Study of Apache Spark in Big Data Processing // International Journal of Computer Science Trends and Technology -2016. - V. 3. - № 4. - P. 93-98.

УДК 004.6+ 004.4

Эволюция понятия и жизненного цикла графов знаний

Апанович З.В.

(Институт систем информатики СО РАН, Новосибирский государственный университет)

В данной работе рассматривается эволюция понятия «граф знаний» с момента возникновения и до текущего момента. Также рассматривается вопрос о том, как эволюция систем, позиционирующих себя как графы знаний, повлияла на определение и жизненный цикл графов знаний.

Ключевые слова: *граф знаний, качество, покрытие, корректность, свежесть, происхождение.*

1. Введение

Первые использования термина «граф знаний» слабо связаны с современной практикой его применения. Этот термин появился еще в 1974 году в контексте определения интерактивного процесса обучения между обучаемым и обучающим. Вершинам графа знаний соответствовали единицы знаний, которые должен изучить обучающийся, а ребра между вершинами соответствовали отношению порядка изучения единиц знаний. [14].

В 1980-х годах исследователи из Нидерландов использовали термин «граф знаний» для формального описания их системы, основанной на извлечении знаний из медицинских и социологических текстов, и собирались постепенно увеличивать количество знаний в этом графе вплоть до построения экспертной системы. [24, 8].

Очередное «громкое» появление термина «граф знаний» под лозунгом «вещи, а не строки» (“things not strings”) было инициировано компанией Google в маркетинговых целях https://www.blog.google/products/search/introducing-knowledge-graph-things-not/. Лозунг подчеркивал полезность устранения неоднозначностей по отношению к сущностям, хранящимся в графе знаний. Смысл этого высказывания состоял в том, что текстовые строки часто неоднозначны, в то время как графы знаний состоят из сущностей, к которым применяется процедура устранения неоднозначностей, так что проще различать сущности, имеющие одинаковое название, но соответствующие разным объектам реального мира. С этого момента термин «граф знаний» стал использоваться по отношению к разным

продуктам, которые отличаются по таким характеристикам как архитектура, цель функционирования, и используемая технология, что затрудняет ответ на вопрос, почему они все позиционируются как «графы знаний».

Работа структурирована следующим образом. В разделах с 1 по 6 приводятся разные определения графов знаний и требования к графам знаний, а также примеры систем, позиционирующих себя как графы знаний, соответствующие этим определениям. На основании рассмотренных примеров предлагается схема жизненного цикла, соответствующего современным графам знаний.

2. Определения, связанные с понятием графа RDF

Поскольку современное использование графов знаний возникло в контексте направления Semantic Web, встречается много публикаций, в которых графом знаний определяют просто как *RDF-граф,* то есть множество триплет в виде (субъект, предикат, объект) [8, 13].

Например, предложение «Человек по имени Сергей Бондарчук снял фильм «Война и мир», снимался в главной роли в фильме «Судьба человека» и был женат на Инне Макаровой и Ирине Скобцевой» можно представить в виде множества триплет, показанных ниже.

dbr:Sergei_Bondarchuk rdf:type dbo:Person.
dbr:Inna Makarova rdf:type dbo:Person.
dbr:Irina Skobtseva. rdf:type dbo:Person.

dbr:Fate_of_a Man rdf:type dbo:Film.

dbr:War_and_Peace rdf:type schema:CreativeWork.

dbr:Sergei_Bondarchuk dbo:spouse dbr:Inna Makarova.

dbr:Sergei_Bondarchuk dbo:spouse dbr:Irina Skobtseva.
dbr:Sergei_Bondarchuk dbo:starring dbr:Fate_of_a Man.
dbr:Sergei_Bondarchuk dbo:director dbr:War_and_Peace.

Такой последовательности триплет соответствует граф, в котором вершинами являются субъекты и объекты триплет, а ребра помечены предикатами этих триплет. Например, показанное выше множество триплет можно изобразить при помощи графа, показанного на Рис. 1.

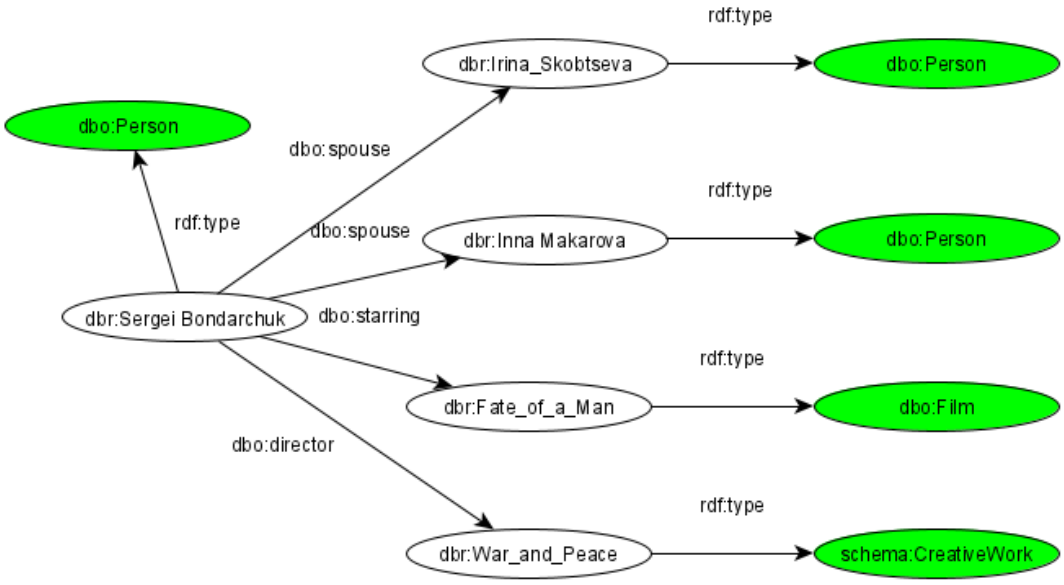


Рис 1. Пример RDF-графа.

Каждая триплета интуитивно представляет *утверждение*. Если граф знаний был построен правильно (со 100% точностью), и при этом данные собирались из достоверных источников данных, то эти «утверждения» можно было бы рассматривать как *факты*.

Такие факты часто представляют с помощью положительных унарных и бинарных логических предикатов логики первого порядка. Например, триплета
dbr:Sergei_Bondarchuk rdf:type dbo:Person
может быть представлена при помощи предиката
Person(Sergei_Bondarchuk), а триплета
dbr:Sergei_Bondarchuk dbo:director dbr:War_and_Peace
при помощи предиката
director (Sergei_Bondarchuk, "War_and_Peace").

Эти же самые факты можно представить при помощи «векторных вложений» "embeddings", которые создают вектора для каждой сущности и каждого отношения. Вектора кодируют латентные свойства сущностей и отношений и обладают свойством, что сходные сущности и сходные отношения будут представлены похожими векторами.

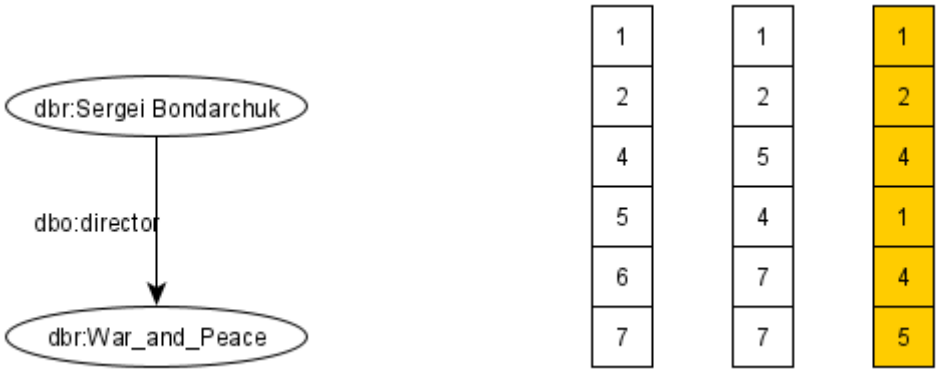


Рис 2. Пример того, что делают методы вложения с одной триплетой графа знаний.

Некоторые определения графов знаний на основе RDF подчеркивают более продвинутый характер графов знаний по сравнению с обычными Связанными данными [25]:

Граф знаний – это структурированный набор данных, собранный из разнородных источников данных, совместимый с моделью данных RDF и имеющий (OWL) онтологию в качестве своей схемы. Граф знаний не обязательно связан с внешними графами знаний; однако сущности в графе знаний обычно имеют информацию о типе, определенном в его онтологии, которая полезна для предоставления контекстной информации о таких сущностях. Графы знаний должны быть надежными, качественными, доступными и ориентированными на информационные услуги для конечного пользователя.

В этом определении в качестве характерной особенности графов знаний подчеркивается необходимость интеграции данных из множественных источников и повышенные требования к качеству контента.

Еще одно весьма популярное определение графов знаний приведено в работе [26]:
Граф знаний
1. в основном описывает сущности реального мира и их взаимосвязи, организованные в виде графа.
2. определяет возможные классы и отношения сущностей в схеме.
3. позволяет потенциально связывать произвольные объекты друг с другом.
4. графы знаний должны покрывать, по крайней мере, большую часть предметных областей, которые существуют в мире, и не должны ограничиваться только одной предметной областью.

Определению [26] удовлетворяют практически все междоменные графы знаний, предназначенные для поиска знаний в Интернете. Среди них DBpedia [17], YAGO [30], Wikidata [33], BabelNet [21], Cyc [18], NELL [6], CaLiGraph [11], VoldemortKG [32].

При этом следует обратить внимание, что все графы знаний из этого списка также обладают значительными объемами.

В таблице 1 показаны некоторые количественные характеристики междоменных графов знаний [12].

Таблица 1 Характеристики некоторых меж-доменных графов знаний [12].

	Экземпляров	**Триплет**	**Классов**	**Отношений**
DBpedia	**5 044 223**	**854 294 312**	**760**	**1355**
YAGO	**6 349 870**	**479 392 870**	**819292**	**77**
Wikidata	**52 252 549**	**732 420 508**	**2 356 259**	**6 236**
BabelNet	**7 735 436**	**178 982`397**	**6 044 564**	**22**
Cyc	**122 441**	**2 229 266**	**116 821**	**148**
NELL	**5 120 588**	**60 594 443**	**1 187**	**440**
CaLiGraph	**7 315918**	**517 099 124**	**755 963**	**271**
Voldemort	**55 861**	**693 428**	**621**	**294**

Значительным объемом контента обладают и наиболее известные графы знаний, представляющие ИТ-индустрию. В таблице 2 показаны некоторые количественные характеристики контента промышленных графов знаний [23].

Таблица 2 Характеристики известных промышленных графов знаний [23]

	Модель данных	Размер графа	Этап разработки
Microsoft	Типы сущностей, отношений и атрибутов графа определены в онтологии	2 миллиарда первичных сущностей, 22 миллиардов фактов	Активно используется в продуктах
Google	Строго типизированные	1 миллиард сущностей, 70 миллиардов	Активно используется в

	сущности, отношения с выводом области определения и области значений	утверждений	продуктах
Facebook	Все атрибуты и отношения структурированы и строго типизированы, опционально проиндексированы для эффективного извлечения, поиска и обхода	50 миллионов сущностей, 500 миллионов утверждений	Активно используется в продуктах
eBay	Сущности и отношения хорошо структурированные и строго типизированные	Ожидается около 100 миллионов продуктов, 1 миллиард триплет	На этапе разработки и запуска
IBM	Сущности и отношения, с котороми ассоциирована информация о свидетельствах, подтверждающих извлеченные факты	Различные размеры Доказанныхдокументов > 100 миллионов Отношений >5 миллиардов Сущностей > 100 миллионов	Активно используется в продуктах и клиентами

Неудивительно, что перечисленные выше системы графов знаний сталкиваются с проблемой *управления графами большого объема*. Проблемы этого измерения графов знаний рассматривались во множестве публикаций в академическом и исследовательском сообществе (например, задача устранения неоднозначности). Тем не менее, в промышленных условиях возникают новые вызовы. Управление масштабом - основная проблема, которая затрагивает операции, напрямую связанные с производительностью и рабочей нагрузкой. Проблема масштаба также проявляется косвенно, так как влияет на другие операции, например, управление быстрыми инкрементными обновлениями для крупномасштабных графов знаний в IBM, или управление согласованностью на большом развивающемся графе знаний в Google [23].

3. Графы знаний, описывающие сущности одного класса

Многие исследователи лишь частично соглашаются с определением графа знаний из [6]. В частности, у многих вызывает возражение тезис о том, что «графы знаний не должны ограничиваться только одной предметной областью". Во-первых, в последнее время наблюдается тенденция по созданию графов знаний, которые собирают информацию обо всех сущностях, принадлежащих одному классу. Например, Amazon и EBay создают графы знаний обо всех продуктах в мире, Google и Apple создали графы знаний обо всех локациях в мире, Центральный банк Италии создал граф знаний обо всех итальянских компаниях, ClaimsKG [31] извлекает утверждения из веб-страниц, занимающихся проверкой фактов, таких как политифакт, и связывает их с другими графами знаний, такими как DBpedia, что также позволяет находить связанные претензии, а EventKG [9] извлекает информацию о месте и времени всех сущностей, относящихся к классу Event (событие).

Во-вторых, многие приложения, созданные для определенных областей, таких как биология, могут считаться графами знаний, если они удовлетворяют другим требованиям графа. К таковым относятся, например, работы, посвященные автоматическому построению графов знаний из текстовых медицинских знаний и медицинских записей [28, 15, 27].

4. Граф знаний как однозначный граф с атрибуцией происхождения

В работе [19], приводится следующее определение графов знаний.

Граф знаний - это «граф, состоящий из множества утверждений (ребер, помеченных отношениями), которые выражаются между сущностями (вершинами графа), где смысл графа закодирован в его структуре, отношения и сущности однозначно идентифицированы, ограниченное множество отношений используется для меток ребер, и граф кодирует происхождение, особенно обоснование и атрибуцию этих утверждений».

В этом определении рассматривается несколько существенных аспектов современного понимания графа знаний.

4.1. Однозначная идентификация сущностей и ребер

Во-первых, авторы подчеркивают, что для того, чтобы утверждения графа знаний были недвусмысленными, они должны состоять из однозначных единиц. То есть, все объекты в

графе знаний, включая типы и отношения, должны быть идентифицированы при помощи глобальных идентификаторов с однозначным обозначением. Одним из примеров такого идентификатора является универсальный идентификатор ресурса (URI), используемый в RDF. К сожалению, требование однозначного идентификатора для каждой сущности связано с давно известной, но до сих пор не решенной проблемой *идентификации сущностей.*

В простейшей форме задача заключается в присвоении уникального нормализованного идентификатора и типа высказыванию или упоминанию объекта. Многие сущности, извлеченные из источников данных автоматически, имеют очень похожие поверхностные формы, например, люди с одинаковыми или похожими именами или фильмы, песни и книги с одинаковыми или похожими названиями. Без правильной привязки и устранения неоднозначности сущности будут неправильно ассоциироваться с неверными фактами и приводить к неверным выводам при последующей обработке.

В случаях, когда управление идентификацией должно выполняться с разнородной базой участников и в масштабе, проблема становится намного сложнее. Например, только в одной Википедии имеется двести Уиллов Смитов, а результат поиска движка Bing для актера Уилла Смита составляется из сто восемь тысяч фактов, взятых с сорока одного сайта. [23]. Кроме того, эффективная система идентификации сущностей также должна расти органично на основе постоянно меняющихся входных данных. Например, компании могут слиться или разделиться, а новые научные открытия могут разбить существующий объект на несколько частей. Если одна компания приобретает другую компанию, изменяется ли идентичность приобретающей компании?

Что касается «ограниченного набора типов отношений», то в контексте системы знаний открытого мира это требование надо понимать как множество основных отношений, которые истинны *независимо от контекста.* Примеры отношений, зависящие от контекста, часто встречаются в DBpedia. Например, в англоязычной DBpedia можно найти следующие две триплеты, касающиеся жен советского режиссера Сергея Бондарчука.

dbr:Sergei_Bondarchuk dbo:spouse dbr:Inna Makarova.

dbr:Sergei_Bondarchuk dbo:spouse dbr:Irina Skobtseva.

Поскольку в DBpedia не указано, что брак Бондарчука с Макаровой продолжался с 1948 по 1956 год, а на Скобцевой он был женат 1959 года вплоть до своей смерти, можно предположить, что у него было две жены одновременно. Чтобы данные утверждения не зависели от контекста, можно расширить эти два утверждения информацией о начале и конце каждого брака. Но описание такой информации требует дополнительных механизмов

таких, как реификация утверждений, либо использование именованных графов, или использование модели графов свойств.

В отличие от DBpedia, информация о браках Сергея Бондарчука отображена более корректно в графе знаний Wikidata. В этом наборе данных у каждого отношения *dbo:spouse* имеется дата начала брака, и дата окончания брака. Поэтому набор данных Wikidata является более качественным источником данных, чем DBpedia.

4.2. Кодирование происхождения утверждений

С точки зрения [19], граф знаний должен быть источником *достоверного* знания, а не просто набором некоторых утверждений. Поэтому каждое важное утверждение графа знаний должно сопровождаться дополнительной информацией о происхождении этого утверждения.

Заметим, что Консорциум World Wide Web разработал стандарт описания происхождения данных в Интернете (PROV) вместе с его представлением в виде онтологии PROV-O [16] и нано-публикаций [10].

Пример. Существуют приложения, где достоверность знаний, представленных в графе знаний, является критичной. Например, в работе [20] описан граф знаний, используемый для поиска новых лекарств против различных разновидностей меланомы, а также новых способов использования известных лекарств. Для решения этой задачи нужны достоверные знания относительно взаимодействия лекарств, белков и заболеваний. При построении графа знаний отбираются только те факты, достоверность которых превышает заданный порог. Поэтому все важные утверждения этого графа знаний снабжены информацией о происхождении утверждений в формате нано-публикации. На Рис. 3 показан пример утверждения о взаимодействии между двумя белками в виде нанопубликации. Представлены три графа. Граф утверждений (утверждение NanoPub 501799_Assertion) утверждает, что взаимодействие (X) имеет тип *sio: DirectInteraction,* имеет целью белок SLC4A8 и участником взаимодействия является белок CA2. Опорный граф (NanoPub 501799 Supporting), утверждает, что граф утверждений был создан в результате эксперимента с понижением (один из многих используемых типов закодированных экспериментов, подкласс *prov: Activity*. Граф атрибуции (NanoPub 501799 Attribution), в свою очередь, заявляет, что это утверждение имеет в качестве первоисточника публикацию (Loiselle et al., 2004), и что взаимодействие было процитировано из BioGrid.

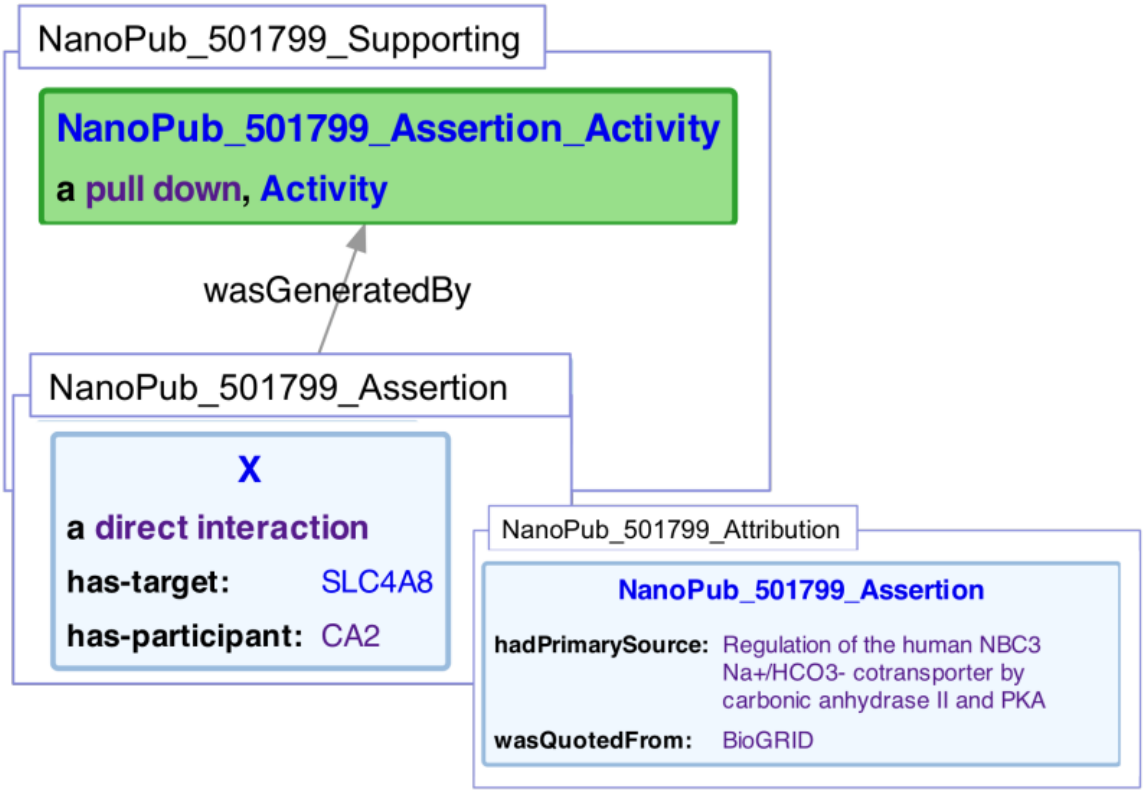


Рис. 3. Пример указания иформации о происхождении утверждения в формате нано-публикации [20].

Для подтверждения своей точки зрения, авторы создали Каталог Графов Знаний (Knowledge Graphs Catalog, KGC) (http: //graphs.whyis.io), в который вошли тридцать семь различных систем, в той или иной мере отвечающим требованиям графа знаний. В частности, системами, удовлетворяющими всем критериям, перечисленным в определении, (структурированный смысл, отсутствие неоднозначностей, отслеживание происхождения, ограниченные отношения (не зависящие от контекста) оказались UniProt KB, Gene Ontology, BioPortal, и сам Knowledge Graph Catalog. Известны и другие графы знаний, хранящие информацию о происхождении. В частности, Facebook [23] использует информацию о происхождении данных для поддержки корректности своего графа знаний. С точки зрения Facebook *корректность* не означает, что граф знаний всегда знает «правильное» значение атрибута, но скорее что всегда можно *объяснить*, *почему* было сделано определенное утверждение. Поэтому он сохраняет *происхождение* всех данных, которые проходят через систему, от сбора данных до слоя сервисов.

5. Графы знаний как системы, позволяющие получать новые знания

В настоящее время имеется значительное количество работ, которые считают, что специфической особенностью графов знаний является не только *способ представления знаний*, но и *способ получения новых знаний.* Так в работе [7] говорится, что специфической особенностью графов знаний, помимо требований большого объема данных, и интеграции множественных источников данных, является использование некоторого механизма порождения новых знаний. Поэтому дается следующее определение графа знаний:

Граф знаний собирает и интегрирует информацию в онтологию и применяет ризонер, чтобы получать новые знания.

В настоящее время имеется множество различных механизмов порождения этого нового знания, основными из которых являются логические методы, основанные на применении правил вывода [29] и статистические методы, основанные на так называемых *векторных вложениях* (embeddings) графов знаний [22], а также различные комбинации этих двух методов.

В сжатой форме подобное определение графа знаний формулируется следующим образом [5]:

«Граф данных, предназначенный для создания новых знаний».

Эти два определения рассматривают не только представление контента графа знаний, но и действия по формированию новых знаний, то есть способы управления графом знаний, что существенно расширяет границы определения понятия графа знаний. Поэтому в [2] представлено три разных взгляда на понятие графа знаний:

- как инструмента представления знаний, где основное внимание уделяется тому, как граф знаний используется для представления некоторой формы знаний;

- системы управления знаниями: основное внимание уделяется управлению системой графа знаний, аналогично тому, как системы управления базами данных играют эту роль для баз данных;

- сервисам приложений знаний: основное внимание уделяется обеспечению уровня приложений поверх графа знаний.

6. Данные, участвующие в создании нового знания

Как только мы признаем, что важной функцией графа знаний является создание новых знаний, важность поддержки *неявных* знаний становится центральным местом для графа

знаний, особенно когда они являются компонентом приложений ИИ предприятия до такой степени, что интенсиональное знание следует рассматривать как часть самого графа знаний.

Например, в финансовых приложениях корпоративных графов знаний множество регулирующих правил и правил функционирования конкретной финансовой области являются существенными. Аналогично, в приложениях логистики знание о том, как взаимодействуют определенные шаги в цепочке поставок, часто более важно, чем обычные данные, лежащие в основе цепочки поставок. В работе [3] утверждается, что «в современных системах, основанных на KG, необходимо учитывать и надлежащим образом обрабатывать богатое представление знаний, чтобы сбалансировать повышенную сложность со многими другими свойствами, включая удобство использования, масштабируемость, производительность и надежность приложения KG». По этой причине [3] предлагает следующее определение графа знаний.

«Полуструктурированная модель данных, состоящая из трех компонентов:

1) базовый экстенсиональный компонент, то есть набор реляционных конструкций для схемы и данных (которые можно эффективно смоделировать в виде графов или их обобщений);

2) интенсиональный компонент, то есть набор правил вывода над конструкциями экстенсиональной компоненты;

3) производный экстенсиональный компонент, который может быть создан в результате применения правил вывода к базовому экстенсиональному компоненту (так называемым процесс «вывода»).

В качестве примера к этому определению рассмотрим пример обнаружения новых связей в графе собственности компаний [1] Центрального банка Италии. Графы собственности компании являются центральными объектами корпоративной экономики и имеют большое значение для центральных банков, финансовых органов и национальных статистических управлений, чтобы решить актуальные проблемы в разных сферах: банковский надзор, оценка кредитоспособности, противодействие отмыванию денег, обнаружение страхового мошенничества, экономические и статистические исследования и многое другое.

Как показано на рисунке 4, на таких графах ключевым понятием является *отношение владения*: узлами являются компании и персоны (черные или синие узлы), а ребра соответствуют отношению собственности (черные сплошные ссылки) и помечены долей акций компании *y*, которыми владеет компания или персона *x*.

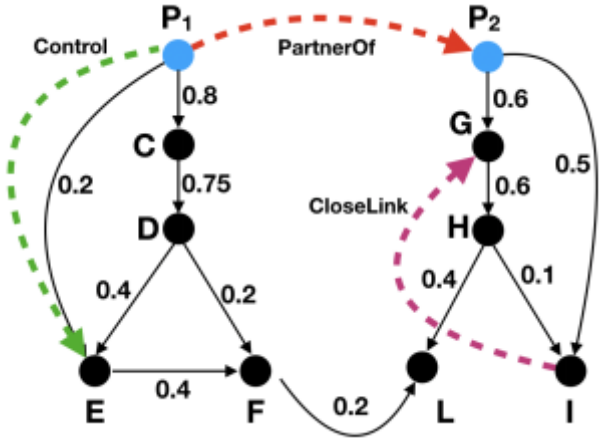


Рис. 4. Пример фрагмента из графа владения компаниями [34]. Ребра, показанные черным цветом, присутствуют изначально в качестве базового контента графа знаний. Ребра зеленого, красного и сиреневого цвета появляются в результате применения комбинированного метода вывода, использующего как логические правила, так и векторные вложения вершин [1].

1) При помощи графов компаний можно решить такую проблему как контроль над компанией. Рассмотрим граф на рисунке 4: Персона P1 управляет компаниями C, D (через C), E (поскольку она контролирует D, которому принадлежит 40% акций E, а также персона P1 напрямую владеет двадцатью процентами акций), и F (через E и D). Точно так же персона P2 контролирует всех своих потомков по графу компаний, кроме L. По-видимому, P1 также не контролирует L. То есть, ребро (P1, E) не принадлежит исходному графу владения компаниями, но эту информацию можно обнаружить при помощи процедуры обнаружения знаний.
2) Второе особенно важное приложение графов компании состоит в оценке риска предоставить конкретную ссуду компании *x*, которая обеспечена под залог, выданный другой «близко связанной» компанией *y*. Например, на рисунке 4, в результате процедуры создания нового знания появляется ребро (I, G), имеющее тип *CloseLink*, означающее, что компании G и I тесно связаны, поскольку персона P2 владеет более чем двадцатью процентами акций обеих компаний.
3) Помимо финансовых отношений, личные или семейные связи позволяют более широкое использование таких графов компаний: обнаружение семейного бизнеса или изучение реального распределения контроля. В примере на рисунке 4 знание того, что персоны P1 и P2 имеют личные связи - например, женаты - позволяет сделать вывод, что на самом деле P1 и P2 вместе управляют компанией L. Скорее всего, они

действуют как единый центр интересов: L на самом деле семейный бизнес, с контролем в руках одной семьи, а P1 и P2 вместе контролируют 60% этой компании.

Таким образом, в исходном графе компаний имеются только ребра, изображающие отношение владения компанией. Ребра, соответствующие отношениям *PartnerOf, Control, CloseLink* «создаются» в результате применения комбинированной процедуры созданий нового знания, комбинирующего применение правил логического вывода к фрагментам исходного графа, полученным на основе векторных вложений вершин графа. В соответствии с определением, данным выше, элементами графа знаний должны быть исходный граф знаний плюс правила вывода, использованные для обнаружения неизвестных связей между вершинами исходного графа, а также эти новые ребра, полученные в результате процедуры «вывода».

Представители фирмы IBM [23] идут еще дальше в понимании того, что должно храниться в графе знаний. Они считают, что *доказательства (или свидетельства*) должны быть примитивами по отношению к системе. Основное звено между реальным миром (которое разработчики часто пытаются моделировать) и структурами данных, содержащими *извлеченное* знание, – это «свидетельства» знания. “Свидетельствами могут быть необработанные документы, базы данных, словари или файлы изображений, текста и видео, из которых знания получены. Когда дело доходит до точных и полезных контекстных запросов во время процесса обнаружения, метаданные и другая связанная информация часто играют важную роль в выводе знаний. Таким образом, критически важно не потерять связь между отношениями, хранящимися в графе, и тем, откуда берутся эти отношения”.

Такое требование вполне сочетается со следующим определением графа знаний [4].

Графы знаний можно представить как сеть всех видов вещей, которые относятся к конкретной предметной области или организации. Они не ограничиваются абстрактными понятиями и отношениями, но могут также содержать экземпляры таких вещей, как документы и наборы данных ".

7. Заключение

В контексте больших промышленных графов знаний таких как графы знаний Microsoft, Google, Facebook, IBM наиболее важными критериями полезности графов знаний являются *покрытие, корректность и свежесть* контента.

Под покрытием понимается наличие в графе всей необходимой информации для предоставления наилучшего сервиса для пользователя. Поэтому ответ на этот вопрос всегда остается отрицательным, а разработчики графов знаний всегда стремятся расширить

множество источников знаний, чтобы увеличить качество покрытия. Заметим, что эта задача является дополнительной по отношению к задаче «завершения графа», которая решается для фиксированного множества источников данных.

Проблема *корректности* связана с вопросом, верна ли информация, предоставляемая пользователю? Действительно ли два источника информации относятся к одному и тому же факту, и что делать, если они противоречат друг другу? Ответы на эти вопросы остаются огромной областью исследования и инвестиций.

Понятие *свежести* контента связано с ответом на вопрос: «Обновлен ли контент»? Возможно, это было правильные когда-то, но устаревшие данные. Свежесть может быть разной для сущностей, которые меняется постоянно (цена акций) и редко меняющихся сущностей (столица страны или название фирмы), с множеством различных вариантов свежести между этими крайними точками. Например, граф знаний Facebook предназначен *для постоянного изменения*. Поэтому граф знаний Facebook - это не единое представление в базе данных, которое обновляется при появлении новой информации. Граф знаний Facebook строится с нуля, из исходников, каждый день, а система сборки идемпотентна и создает полный граф в конце сборки [23].

Таким образом, рассмотрев все требования к представлениям знаний в современных графах знаний, можно представить обобщенную картину жизненного цикла современного графа знаний, как процесс постоянного расширения внешний источников данных, алгоритмов, позволяющих создавать новые знания из этого расширяющегося множества источников данных, и, наконец, постоянно расширяющееся множество приложений, создаваемых над графами знаний. См. рисунок 5.

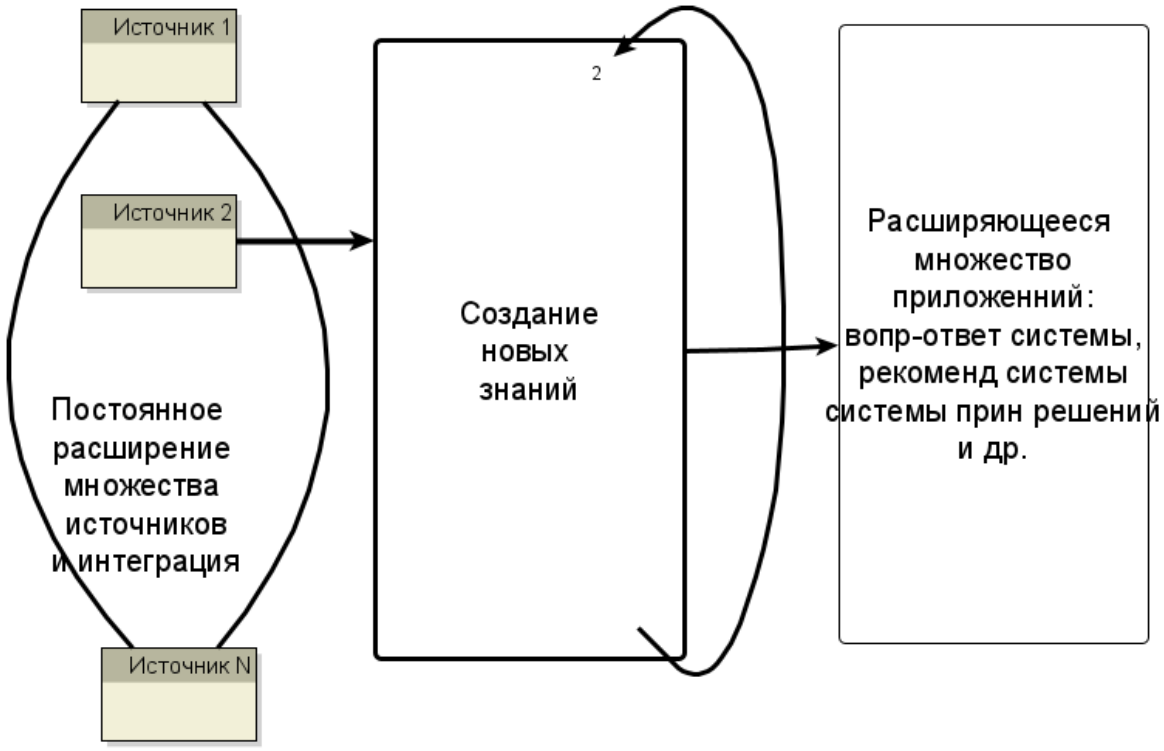


Рис. 5 Жизненный цикл графов знаний.

Если еще несколько лет назад основным приложением графов знаний считался семантический поиск, то сейчас в этот список входят вопросно-ответные системы, рекомендательные системы, системы принятия решений и многие другие. Более того, именно возможность обнаружения новых знаний и создания на их основе новых приложений является основным поводом для разработки корпоративных графов знаний.

Список литературы

1. Atzeni P., Bellomarini L., Iezzi M., Sallinger E., Vlad A. Weaving Enterprise Knowledge Graphs: The Case of Company Ownership Graphs https://openproceedings.org/2020/conf/edbt/paper_334.pdf
2. Bellomarini L., Sallinger E., Vahdati S. Chapter 2 Knowledge Graphs: The Layered Perspective/ / Janev V., Graux D., Jabeen H., Sallinger E. (eds) Knowledge Graphs and Big Data Processing. Lecture Notes in Computer Science. 2020. vol 12072. Springer, Cham. https://doi.org/10.1007/978-3-030-53199-7_2
3. Bellomarini, L., Fakhoury, D., Gottlob, G., Sallinger, E.: Knowledge graphs and enterprise AI: the promise of an enabling technology// 2019. IEEE 35th International Conference on Data Engineering (ICDE), P. 26–37.
4. Blumauer, A.: From taxonomies over ontologies to knowledge graphs (2014) https://semantic-web.com/2014/07/15/from-taxonomies-over-ontologies-to-knowledge-graphs/

5. Bonatti, P.A., Decker, S., Polleres, A., Presutti, V.: Knowledge graphs: new directions for knowledge representation on the semantic web (Dagstuhl Seminar 18371)// Dagstuhl Rep. 2019. Vol. 8, № 9. P. 29–111.
6. Carlson A., Betteridge J.,. Wang R. C, Hruschka E. R Jr, Mitchell T. M. Coupled semi-supervised learning for information extraction// *Proceedings of the third ACM international conference on Web search and data mining* 2010. P. 101–110.
7. Ehrlinger L., Woß W. Towards a definition of knowledge graphs//SEMANTiCS (Posters, Demos, SuCCESS), 2016. № 48.Ernst P., Siu A., Weikum G., KnowLife: a versatile approach for constructing a large knowledge graph for biomedical sciences, *BMC bioinformatics vol.* 2015. Vol. **16**, № 1. P. 157.
8. Färber M., Bartscherer F., Menne C., Rettinger A., Linked data quality of dbpedia, freebase, opencyc, wikidata, and yago// *Semantic Web.* 2016, P. 1–53.
9. Gottschalk S, Demidova E. EventKG: A Multilingual Event-Centric Temporal Knowledge Graph https://arxiv.org/pdf/1804.04526.pdf
10. Groth P., Gibson A., Velterop J., "The anatomy of a nanopublication," *Information Services & Use*, 2010. vol. 30, N. 1-2, P. 51–56.
11. Heist N., Paulheim H.. Entity extraction from wikipedia list pages// *Extended Semantic Web Conference*, 2020.
12. Heist N., Hertling S., Ringler D., Paulheim H. Knowledge Graphs on the Web – an Overview https://arxiv.org/pdf/2003.00719.pdf 2020
13. Huang Z., Yang J., Harmelen F. van, Hu Q., Constructing disease-centric knowledge graphs: a case study for depression (short version), in: *Conference on Artificial Intelligence in Medicine in Europe*, Springer, 2017, P. 48–52.
14. James P.. Knowledge Graphs. // Linguistic Instruments in Knowledge Engineering, Elsevier Science Publishers B.V.,.1992. P. 97-117.
15. Lamurias A., Ferreira J.D., Clarke L.A., Couto F.M., Generating a Tolerogenic cell Therapy Knowledge graph from literature, *Frontiers in immunology* 2017. № 8, P. 1656.
16. Lebo T., Sahoo S., McGuinness D., Belhajjame K., Cheney J., Corsar D., Garijo D.,. Soiland-Reyes S, Zednik S., Zhao J., "PROV-O: The prov ontology," *W3C recommendation*, 2013.
17. Lehmann J., Isele R., Jakob M., Jentzsch A., Kontokostas D., Mendes P. N., Hellmann S., Morsey M., van Kleef P., Auer S., Bizer C.. DBpedia – A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia// *Semantic Web Journal*. 2013. Vol. 6. № 2.
18. Lenat D. B.. CYC: A large-scale investment in knowledge infrastructure *Communications of the ACM*, 1995. Vol. 38 №1. P. 33–38.

19. McCusker J. P., Erickson J. S., Chastain K., Rashid S., Weerawarana R., Bax M., McGuinness D. L. What is a knowledge graph http://www.semantic-web-journal.net/system/files/swj1954.pdf, 2018
20. McCusker J.P., Dumontier M., Yan R., He S., Dordick J.S., McGuinness D.LFinding melanoma drugs through a probabilistic knowledge graph.// PeerJ Computer Science . (2017) 3:e106 https://doi.org/10.7717/peerj-cs.106
21. Navigli R., Ponzetto S. P. Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network// *Artificial Intelligence*. 2012. № 193 P. 217–250.
22. Nickel, M., Murphy, K., Tresp, V., Gabrilovich, E. A review of relational machine learning for knowledge graphs.//IEEE . 2016. vol. 104, 1. № . P. 11-33.
23. Noy N., Gao Y., Jain A., Narayanan A., Patterson A., Taylor J.. Industry-scale knowledge graphs: Lessons and challenges//*Communications of the ACM* . 2019. Vol. 62. № 8. P. 36–43.
24. Nurdiati S., Hoede C. 25 Years Development of Knowledge Graph Theory: The Results and the Challenge, September 2008.
25. Pan J. Z., Vetere G., Gomez-Perez J. M., Wu H. Editors. Exploiting Linked Data and Knowledge Graphs in Large Organizations Springer, 2017.
26. Paulheim H.. Knowledge graph refinement: A survey of approaches and evaluation methods// *Semantic Web*. 2017, Vol. 8. №3 . P. 489–508,
27. Rotmensch M., Halpern Y., Tlimat A., Horng S., Sontag D., Learning a health knowledge graph from electronic medical records// *Scientific reports.* 2017. Vol. **7**, № 1. P. 5994.
28. Shi L., Li S., Yang X., Pan J. Qi, G., Zhou B., Semantic health knowledge graph: Semantic integration of heterogeneous medical knowledge and services, *BioMed Research International* 2017.
29. Stepanova D., Gad-Elrab M. H., Thinh V. Ho Rule Induction and Reasoning over Knowledge Graphs https://people.mpi-inf.mpg.de/~dstepano/conferences/RW2018/paper/RW2018paper.pdf
30. Suchanek F. M., Kasneci G., Weikum G.. YAGO: A Core of Semantic Knowledge Unifying WordNet and Wikipedia. // *16th international conference on World Wide Web*. 2007. P. 697–706,
31. Tchechmedjiev A., Fafalios P.,. Boland K, Gasquet M., Zloch M., ¨Zapilko B., Dietze S., Todorov K.. ClaimsKG: A knowledge graph of fact-checked claims. In International Semantic Web Conference, Springer, 2019. P. 309–324.
32. Tonon A., Felder V., Difallah D. E., Cudre-Mauroux P. Voldemortkg: ´Mapping schema. org and web entities to linked open data// *International Semantic Web Conference*, Springer, 2016. P. 220–228.
33. Vrandecic D., Krotzsch M.. Wikidata: a Free Collaborative Knowledge Base.// *Communications of the ACM* 2014. Vol. 57, № 10. P. 78–85.

УДК 004.05

Верификация предикатной программы пирамидальной сортировки с применением обратных трансформаций

Шелехов В.И. (Институт систем информатики СО РАН,
Новосибирский государственный университет)

Проводится дедуктивная верификация алгоритма классической пирамидальной сортировки Дж. Вильямса, реализованного программой sort на языке Си в библиотеке ОС Linux. Сортировка реализуется для объектов произвольного типа. Чтобы упростить верификацию, применяются нетривиальные трансформации, заменяющие арифметические операции с указателями явными элементами сортируемого массива. Программа преобразуется на язык предикатного программирования. Конструируются спецификации предикатной программы. Дедуктивная верификация в системах Why3 и Coq оказалась сложной и трудоемкой.

Ключевые слова: *дедуктивная верификация, трансформации программ, функциональное программирование, предикатное программирование, неинтерпретированный тип.*

1. Введение

Исходной задачей является дедуктивная верификация программы sort на языке Си из библиотеки ядра ОС Linux. Используется алгоритм классической пирамидальной сортировки Дж. Вильямса [25]. И хотя это наиболее простой алгоритм в классе алгоритмов пирамидальной сортировки, его дедуктивная верификация оказывается нетривиальной.

Сортировка реализуется для объектов произвольного типа и произвольного размера. В языке Си такие объекты представляются указателями общего вида void*. Операции с объектами реализуются функциями, доступными через параметры программы сортировки. Имеются две таких операции: сравнение и обмен пары элементов. Массивы произвольного размера и программы, подставляемые параметрами, существенно затрудняют верификацию программы набором инструментов FramaC – Why3 [2, 16]. Есть еще одна особенность, принципиально затрудняющая верификацию: для вычисления указателей применяется оптимизация уменьшения силы операций с заменой в цикле умножения на сложение.

Дедуктивная верификация намного проще и быстрее для функциональных программ, чем для аналогичных императивных программ. Этот факт отмечается разными исследователями. Причина сложности императивных программ в том, что указатели, конструкции необходимые для оптимизации программ, существенно усложняют логику императивных программ. Для упрощения императивных программ применяются трансформации, устраняющие указатели в императивной программе [9]. Операции с указателями заменяются эквивалентными действиями без указателей. Далее к полученной программе применяются трансформации, превращающие ее в эквивалентную предикатную программу.

В настоящей работе применяется метод обратной трансформации [9] от исходной библиотечной программы sort.c к эквивалентной предикатной программе. Разрабатываются спецификации для полученной предикатной программы. Далее строятся формулы корректности программы относительно спецификации применением системы правил [11]. Совокупность формул корректности вместе с описаниями типов и переменных оформляется в виде набора теорий. Эти теории транслируются на язык спецификаций why3 [24]. Далее в системах дедуктивной верификации Why3[24] и Coq [17] реализуется процесс доказательства формул корректности.

Ранее в 2012г. дедуктивная верификация проводилась для трех алгоритмов пирамидальной сортировки [12]: классического алгоритма Дж. Вильямса [25], алгоритма Флойда [20] и улучшенного алгоритма, [23] бывшего тогда самым быстрым алгоритмом сортировки. В 2019г. дедуктивная верификация той же программы sort проводилась с применением прямых трансформаций, однако не была завершена.

Во втором разделе дается краткое описание языка предикатного программирования. Метод дедуктивной верификации описывается в третьем разделе. В четвертом разделе описывается обратная трансформация исходной программы sort с получением предикатной программы пирамидальной сортировки. В следующем разделе описывается процесс спецификации предикатной программы. Особенности процесса дедуктивной верификации предикатной программы в системах Why3[24] и Coq [17] описывается в шестом разделе. Далее обзор других работ по дедуктивной верификации программы heapsort. В заключении суммируются результаты работы. В Приложении 1 код исходной программы sort на языке Си из библиотеки ОС Linux. В Приложении 3 приведены три теории на языке Why3 для доказательства формул корректности на момент завершения работы по верификации. Доступна полная версия текста настоящей работы: https://persons.iis.nsk.su/files/persons/pages/sort9.pdf.

2. Язык предикатного программирования

Полная предикатная программа состоит из набора рекурсивных *предикатных программ* на языке P [4] следующего вида:

<имя программы>(<описания аргументов>**:** <описания результатов>)
pre <предусловие>
post <постусловие>
measure <выражение>
{ <оператор> }

Предусловие и постусловие являются формулами на языке исчисления предикатов. Они обязательны при дедуктивной верификации [7, 8, 12, 15, 22]. Мера задается только для рекурсивных программ и используется для верификации.

Ниже представлены основные конструкции языка P: оператор присваивания, блок (оператор суперпозиции), параллельный оператор, условный оператор, вызов программы и описание переменных, используемое для аргументов, результатов и локальных переменных.

<переменная> = <выражение>
{<оператор1>; <оператор2>}
<оператор1> || <оператор2>
if (<логическое выражение>) <оператор1> **else** <оператор2>
<имя программы>(<список аргументов>**:** <список результатов>)
<тип> <пробел> <список имен переменных>

Всякая переменная характеризуется *типом* – множеством допустимых значений. *Описание типа* **type** T(p) = D с возможными параметрами p связывает имя типа T с его изображением D. Типы **bool**, **int**, **real** и **char** являются *примитивными*. Значением типа **array**(T_e, T_i) является *массив* с *элементами массива* типа T_e и *индексами* конечного типа T_i. Тип массива является предикатным типом, его значения (массивы) являются тотальными и однозначными предикатами.

Пусть E(x) – логическое выражение. Тип **subtype**(T x**:** E(x)) определяет *подтип* типа T при истинном предикате E(x), т.е. множество {x∈T | E(x)}. Определенный в языке P тип целых чисел **nat** представляется описанием:

type nat = **subtype**(**int** x: x ≥ 0) .

Допускаются подтипы, параметризуемые переменными. Примером является тип *диапазона* целых чисел:

type Diap(**nat** n) = **subtype**(**int** x: x ≥ 1 & x ≤ n) .

В языке P для изображения типа диапазона используется конструкция 1..n.

Описания типов переменных являются частью спецификации программы. Описание переменной T x есть утверждение x∈T, которое становится частью предусловия, если x – аргумент предикатной программы, или частью постусловия, если x – результат программы. При этом утверждение x∈T обычно не пишется в составе предусловия или постусловия, хотя предполагается.

В языке предикатного программирования P [4] нет указателей, серьезно усложняющих программу. Вместо указателей используются объекты алгебраических типов: списки и деревья. Предикатная программа существенно проще в сравнении с императивной программой, реализующей тот же алгоритм. Эффективность предикатных программ достигается применением *оптимизирующих трансформаций* [3]. Они определяют отличную от классической оптимизацию среднего уровня с переводом предикатной программы в эффективную императивную программу.

Базовыми трансформациями являются:

- склеивание переменных, реализующее замену нескольких переменных одной;
- замена хвостовой рекурсии циклом;
- открытая подстановка программы на место ее вызова;
- кодирование объектов алгебраических типов (списков и деревьев) при помощи массивов и указателей.

3. Дедуктивная верификация

Предикатная программа относится к классу *программ-функций* [10]. Программа-функция должна всегда **нормально завершаться** с получением результата, поскольку бесконечно работающая и невзаимодействующая программа бесполезна.

Спецификацией предикатной программы H(x**:** y) являются два предиката: *предусловие* P(x) и *постусловие* Q(x, y). Спецификация записывается в виде: [P(x), Q(x, y)].

Для языка $\mathbf{P_0}$ построена формальная операционная семантика $\mathcal{R}(H)(x, y)$ и доказано тождество $\mathcal{R}(H) = H$ [13]. На базе языка $\mathbf{P_0}$ последовательным расширением и сохранением тождества $\mathcal{R}(H) = H$ построен язык предикатного программирования **P** [4].

Тотальная корректность программы относительно спецификации определяется формулой:

$$H(x\mathbf{:}\ y)\ \mathbf{corr}\ [P(x), Q(x, y)] \cong \forall x.\ P(x) \Rightarrow [\forall y.\ H(x\mathbf{:}\ y) \Rightarrow Q(x, y)\ ]\ \&\ \exists y.\ H(x\mathbf{:}\ y)$$

Формулу тотальной корректности будем представлять в виде правила **COR**:

$$\textbf{COR}: \frac{\begin{array}{l}\forall x,y.\ P(x)\ \&\ H(x\textbf{:}\ y) \Rightarrow Q(x,\ y);\\ \forall x.\ P(x) \Rightarrow \exists y.\ H(x\textbf{:}\ y)\end{array}}{H(x\textbf{:}\ y)\ \textbf{corr}\ [P(x),\ Q(x,\ y)]}$$

Для базисных операторов (параллельного, условного и суперпозиции) разработана универсальная система правил доказательства их корректности [6, 11], в том числе и при наличии рекурсивных вызовов, существенно упрощающая процесс доказательства по сравнению с исходной формулой тотальной корректности. Корректность правил доказана [4] в системе PVS. В системе предикатного программирования реализован генератор формул корректности программы. Часть формул доказывается автоматически SMT-решателем CVC4. Оставшаяся часть формул генерируется для системы интерактивного доказательства PVS [21]. Данный метод опробован для дедуктивной верификации более сотни программ [7, 8, 12, 15, 22].

Предположим, что наборы переменных x, y и z не пересекаются, а x может быть пустым. Ниже приведены некоторые правила доказательства корректности операторов.

$$\textbf{QP}: \frac{B(x\textbf{:}\ y)\ \textbf{corr}\ [P(x),\ Q(x,\ y)];\ C(x\textbf{:}\ z)\ \textbf{corr}\ [P(x),\ R(x,\ z)];}{\{B(x\textbf{:}\ y)\ ||\ C(x\textbf{:}\ z)\}\ \textbf{corr}\ [P(x),\ Q(x,\ y)\ \&\ R(x,\ z)]}$$

$$\textbf{QC}: \frac{\begin{array}{l}B(x\textbf{:}\ y)\ \textbf{corr}\ [P(x)\ \&\ E(x),\ Q(x,\ y)];\\ C(x\textbf{:}\ z)\ \textbf{corr}\ [P(x)\ \&\ \neg E(x),\ Q(x,\ y)]\end{array}}{\{\textbf{if}\ (E(x))\ B(x\textbf{:}\ y)\ \textbf{else}\ C(x\textbf{:}\ y)\}\ \textbf{corr}\ [P(x),\ Q(x,\ y)]}$$

Далее следует правило для частного случая оператора суперпозиции, соответствующего *сведению к более общей задаче* C(x, z: y).

$$\textbf{RB}: \frac{\begin{array}{l}\forall z\ C(x,\ z\textbf{:}\ y)\ \textbf{corr*}\ [P_C(x,\ z),\ Q_C(x,\ y)];\\ P(x) \Rightarrow P_B(x)\ \&\ P^*_C(x,\ B(x));\\ \forall y\ (\ P(x)\ \&\ Q_C(B(x),\ y) \Rightarrow Q(x,\ y)\ );\end{array}}{C(x,\ B(x)\textbf{:}\ y)\ \textbf{corr}\ [P(x),\ Q(x,\ y)]}$$

Запись вида z = B(x) является эквивалентом B(x: z). Истинность трех посылок правила **RB** гарантирует корректность следующей программы:

$$H(x\textbf{:}\ y)\ \textbf{pre}\ P(x)\ \textbf{post}\ Q(x,\ y)\ \{\ C(x,\ B(x)\textbf{:}\ y)\ \}$$

В случае рекурсивного вызова C(x, B(x): y) обозначение **corr*** означает, что первая посылка опускается, а $P^*_C(x, B(x))$ заменяется на $P_C(x, B(x))\ \&\ m(x) < m(v)$. Здесь m – натуральная функция *меры*, строго убывающая на аргументах рекурсивных вызовов, а v обозначает аргументы рекурсивной программы C.

4. Обратная трансформация программы пирамидальной сортировки

4.1. Постановка задачи

В библиотеке ядра ОС Linux имеется программа sort на языке Си, реализующая сортировку объектов произвольного типа и произвольного размера. Используется алгоритм классической пирамидальной сортировке Дж.Вильямса [25]. Программа sort приведена в Приложении 1. Представим заголовок программы:

```
void sort(void *base, size_t num, size_t size,
        int (*cmp_func)(const void *, const void *),
        void (*swap_func)(void *, void *, int size))
```

Здесь base – указатель сортируемого массива; num – число элементов массива; size - размер элемента в байтах; cmp_func – указатель функции сравнения двух элементов; swap_func – указатель функции обмена двух элементов или NULL. В случае swap_func = NULL обмен элементов реализуется одной из функций в составе программы sort.

Функция cmp_func(pa, pb), где pa и pb – указатели на элементы a и b, определяется следующим образом.

```
cmp_func(pa, pb) > 0 -  →  a > b
cmp_func(pa, pb) <= 0 -→  a <= b
cmp_func(pa, pb) = 0 -  →  a = b
```

Ниже представлен код программы sort. Данная программа sort ранее входила в состав библиотеки ядра ОС Linux до версии 5.1. В последующих версиях ОС Linux программа sort заменена другим более быстрым алгоритмом пирамидальной сортировки.

```
void sort(void *base, size_t num, size_t size,
          int (*cmp_func)(const void *, const void *),
          void (*swap_func)(void *, void *, int size))
{
        /* pre-scale counters for performance */
        int i = (num/2 - 1) * size, n = num * size, c, r;
.....
        /* heapify */
        for ( ; i >= 0; i -= size) {
                for (r = i; r * 2 + size < n; r  = c) {
                        c = r * 2 + size;
                        if (c < n - size &&
                                        cmp_func(base + c, base + c + size) < 0)
                                c += size;
                        if (cmp_func(base + r, base + c) >= 0)
                                break;
                        swap_func(base + r, base + c, size);
                }
        }
        /* sort */
        for (i = n - size; i > 0; i -= size) {
                swap_func(base, base + i, size);
                for (r = 0; r * 2 + size < i; r = c) {
                        c = r * 2 + size;
                        if (c < i - size &&
                                        cmp_func(base + c, base + c + size) < 0)
                                c += size;
                        if (cmp_func(base + r, base + c) >= 0)
                                break;
                        swap_func(base + r, base + c, size);
                }
        }
}
```

Отметим, что в коде программы опущена инициация значения функции swap_func в случае swap_func = NULL. Эту инициацию и используемые подпрограммы обмена элементов можно верифицировать независимо. Далее будем считать, что функция swap_func задана параметром программы sort.

Требуется трансформировать данную программу в эквивалентную предикатную программу и провести ее дедуктивную верификацию. Здесь применяются обратные трансформации по сравнению с обычными (прямыми) оптимизирующими трансформациями, используемыми в предикатном программировании.

4.2. Устранение указателей

Целью первой стадии обратных трансформаций является устранение указателей. Операции с указателями заменяются эквивалентными действиями без указателей. Например, адресное вычисление base + c заменяется явным элементом массива base[c/size].

Особенность программы sort.c в том, что для адресных вычислений в итоговой программе на языке Си проведена оптимизация уменьшения силы операций, в результате которой адресные выражения вида base + c * size заменены на base + c посредством изменения масштаба переменных, входящих в c. В такой ситуации для проведения трансформации программы sort необходимо сначала провести трансформацию, обратную уменьшению силы операций, которая заменила бы base + c на base + c * size. Данная трансформация реализуется крайне редко, лишь когда оптимизацию «уменьшение силы операций» применяет программист. Обычно такая оптимизация реализуется при оптимизирующей трансляции.

В случае, когда требуется провести открытую подстановку кода функции на место ее вызова, используется описатель **inline**, гарантирующий проведение данной оптимизации при трансляции. Однако по непонятным причинам, открытая подстановка в программе sort.c проведена явно программистом. Это вынуждает нас провести обратную трансформацию запроцедуривания. Тела внутренних циклов по r почти совпадают. Два цикла по r похожи и отличаются лишь в паре позиций. Введем параметры k и m для этих позиций и определим программу siftDown, телом которой является цикл по r.

Сначала реализуется трансформация запроцедуривания. Далее применяется трансформация, обратная оптимизации уменьшения силы операций, реализующей замену в циклах умножение на сложение. В программе переменные i, r, n и c пересчитываются на значение size в каждом из двух циклов. Проводятся замены переменных:

```
i → ii*size;
r → rr*size;
c → cc*size;
n → nn*size;
```

При этом в циклах параметр i заменяется на ii, в результате чего пересчет на size заменяется пересчетом на единицу. Внутренние циклы по r заменяются циклами по переменной rr. Применение двух трансформаций преобразует программу к следующему виду.

```
        int ii = (num/2 - 1), nn = num, cc, rr;
        for ( ; ii >= 0; ii -= 1) {
                siftDown(base, ii, nn)
        }
        for (ii = nn - 1; i > 0; ii -= 1) {
                swap_func(base, base + ii*size, size);
                siftDown(base, 0, ii)
        }

void siftDown(void *base, const int k, m) {
for (int rr = k; rr *2 + 1 < m; rr  = cc) {
                cc = rr * 2 + 1;
                if (cc < m - 1 &&
                                cmp_func(base + cc*size, base + cc*size + size) < 0)
                        cc += 1;
                if (cmp_func(base + rr*size, base + cc*size) >= 0)
                        break;
                swap_func(base + rr*size, base + cc*size, size);
}
}
```

Будет удобным провести следующие обратные замены переменных:

ii → i; rr → r; cc → c; nn → num;

Доступ в программе к некоторому элементу m массива base реализуется указателем base + m*size. Заменим это указатель на base[m]. Далее применяются трансформации вида:

```
base + p*size                  →  base[p]
swap_func(base+r*size, base+c*size, size)   →   swap(base, r, c)
cmp_func(base+r*size, base+c*size)   →  cmp(base[r], base[c])
```

Получим следующую программу:

```
        int i = (num/2 - 1), c, r;
        for ( ; i >= 0; i -= 1) {
                siftDown(base, i, num);
        }
        for (i = num - 1; i > 0; i -= 1) {
                swap(base, 0, i);
                siftDown(base, 0, i);
        }
```

```
void siftDown(void *base, const int k, m) {
for (int r = k; r *2 + 1 < m; r  = c) {
                c = r * 2 + 1;
                if (c < m - 1 && cmp(base[c], base[c+1]) < 0)
                        c += 1;
                if (cmp(base[r], base[c]) >= 0)
                        break;
                swap(base, r, c);
}
}
```

Оформим полученную программу. Константные параметры программы sort определим глобальными переменными. Введем тип T для элементов сортируемого массива.

```
size_t num, size;
type T;
type Ar = array (T, int);
int cmp(const T, const T);
void swap (Ar, const int, const int);
void sort(Ar base) {
        int i = (num/2 - 1), c, r;
        for ( ; i >= 0; i -= 1) {
                siftDown(base, i, num);
        }
        for (i = num - 1; i > 0; i -= 1) {
                swap(base, 0, i);
                siftDown(base, 0, i);
        }
}
void siftDown(Ar base, const int k, m) {
for (int r = k; r*2 + 1 < m; r  = c) {
                c = r * 2 + 1;
                if (c < m - 1 && cmp(base[c], base[c+1]) < 0)
                        c += 1;
                if (cmp(base[r], base[c]) >= 0)
                        break;
                swap(base, r, c);

}
}
```

4.3. Трансформация в предикатную программу

Определим типы и глобальные переменные.

```
nat num, size;
type T;
nat Di = 0..num-1
type Ar = array (T, Di);
cmp(T, T: int);
swap(Ar, Di, Di: Ar);
```

Заменим циклы **for** на циклы вида **loop**.

```
void sort(Ar base: Ar base') {
        int i = (num/2 - 1), c, r;
        loop {
                if (i < 0) break;
                siftDown(base, i, num: Ar base1);
                i -= 1;
        }
        i = num - 1;
        loop {
                if (i <= 0) break;
                swap(base1, 0, i: base2);
                siftDown(base2, 0, i: base');
        i -= 1;
        }
}

void siftDown(Ar base, const int k, m: Ar base') {
int r = k;
loop {
                if (r*2 + 1 >= m) break;
                int c = r *2 + 1;
                if (c < m - 1 && cmp(base[c], base[c+1]) < 0) c += 1;
                if (cmp(base[r], base[c]) >= 0)  break;
                swap(base, r, c: base);
                r  = c;
        }
}
```

Штрих в имени base' предполагает склеивание переменных base и base' в реализации.

Три цикла преобразуются в рекурсивные программы.

```
heapify(Ar base, int i: Ar base') {
        if (i < 0) base' = base
        else { SiftDown(base, i, num: Ar base1);  heapify(base1, i - 1: base') }
}
sorting(Ar base, int i: Ar base') {
        if (i <= 0) base' = base
        else {  swap(base, 0, i: Ar base1);
                siftDown(base1, 0, i: Ar base2);
                sorting(base2 i - 1: base')
        }
```

```
}
siftDown(Ar base, int r, m: Ar base') {
        int c = r *2 + 1;
        if (c >= m) base' = base
        else {   if (c < m - 1 && cmp(base[c], base[c+1]) < 0) c1 = c+1 else c1 = c;
                  if (cmp(base[r], base[c1]) >= 0) base' = base
                  else { swap(base, r, c1: Ar base1); siftDown(base1, c1, m: base')};
                }
}
```

В итоге получим:

```
sort(Ar base: Ar base') {
        heapify(base, num/2 - 1: Ar base1);
        sorting(base1, num - 1: base')
}
```

Полная предикатная программа, состоящая из программ sort, heapify, sorting и siftDown, в точности соответствует исходной библиотечной программе sort на языке Си.

5. Спецификация предикатной программы sort

Воспроизведем глобальные описания программы sort.

```
nat num;
type T;
nat Di = 0..num-1;
type Ar = array (T, Di);
cmp(T, T: int);
swap(Ar a, Di j, k: Ar a') post exchange(a, a', j, k);
Ar base;
```

Предикат exchange определен в библиотеке Array системы верификации Why3 [16]. Исходный сортируемый массив base определен здесь как глобальный.

Определение и свойства функции cmp для сравнения элементов задаются в виде теории:

```
theory Compare {
   type CMP= predicate(T, T: int)
   CMP cmp_func;
   axiom Refl :   ∀ x: T. cmp_func(x, x) = 0
   axiom Simm :   ∀ T x, y. cmp_func(x, y) = - cmp_func(y, x);
   axiom TotalLe : ∀ T x, y. cmp_func(x, y) <=0 or cmp_func(y, x) <=0;
   axiom TransLe : ∀ T x, y, z. cmp_func(x, y)<=0 & cmp_func(y, z)<=0 ⇒ cmp_func(x,z)<=0;
}
```

Здесь приведены именно те свойства функции сравнения, которые были использованы при доказательствах. Это значит, что в любом вызове программы sort параметр-функция, подставляемая на место cmp_func, должна удовлетворять перечисленным свойствам. Отметим, что аксиома антисимметричности не использовалась.

5.1. Спецификация главной программы

Определим спецификацию программы sort.

```
sort( : Ar base') post sorted(base') & perm(base, base');
{  heapify(base, num/2 - 1: Ar base1);
   sorting(base1, num - 1: base')
}
```

Спецификация определяет, что массив base' должен быть сортирован и получен перестановкой исходного массива base. Свойство сортированности определяет упорядоченность элементов:

```
formula sorted(Ar a) =  ∀ k, j= 0..num-1. k < j ⇒  cmp(a[k], a[j]) <= 0;
formula sortedP(Ar a, nat m) =  ∀ k, j= m..num-1. k < j ⇒  cmp(a[k], a[j]) <= 0;
```

Вторая формула определяет упорядоченность части массива a от m до конца массива.

```
formula perm(Ar a, b) = permut_all(a, b);
formula permE(Ar a, b, nat m, n) = permut_sub(a, b, r, m);
```

Предикаты *перестановочности* заимствованы из Why3[16]. Предикат permut_sub определяет перестановочность на отрезке от m до n-1 и равенство массивов везде вне этого отрезка.

5.2. Двоичная куча

Существует простой, эффективный и компактный способ представления двоичного дерева внутри массива a. Вершинами дерева являются элементы a[0], a[1], …, a[m-1], где m ≤ num. На рис.1 дается пример представления дерева для m = 8.

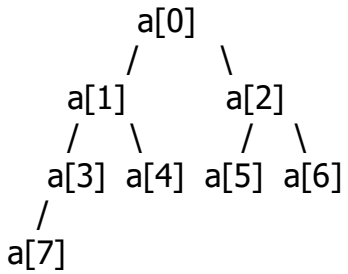


Рис.1. Представление двоичного дерева внутри массива

Родитель вершины с индексом j имеет индекс (j-1) / 2, а левая и правая дочерние вершины имеют индексы 2 j + 1 и 2 j + 2, соответственно. Определения функций для правого и левого потомка и родителя вершины a[j] даются ниже.

```
formula left(nat j : nat) = j * 2 + 1;
formula right(nat j : nat) = j * 2 + 2;
formula father(nat j : int) = (j-1) / 2;
```

Здесь «/» – операция целочисленного деления.

Двоичная максимальная куча (или пирамида) есть двоичное дерево, представленное внутри массива, в котором значение каждой вершины не меньше значений ее потомков.

```
formula heap(Ar a) = ∀ nat j = 0 .. num-1. heapJ(a, num, j)
formula heapJ(Ar a, nat j, m) =    (left(j) < m ⇒ cmp(a[j], a[left(j)])>=0)) &
                                   (right(j) < m ⇒ cmp(a[j], a[right(j)]>=0)) .
```

Предикат heapJ определяет, что вершина j обладает *свойством кучи*.

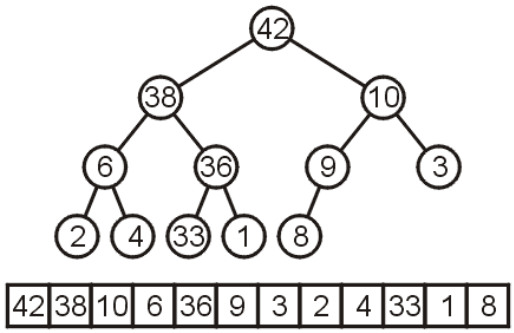


Рис.2. Пример двоичной максимальной кучи при num = 12

Используется также *обобщенная двоичная куча*, построенная на части массива a для элементов в диапазоне индексов от i до m-1 и определяемая формулой:

```
formula heap(Ar a, nat i, m) = ∀ nat j = i .. m-1. heapJ(a, j, m);
```

Дочерние вершины определяются функциями left и right как для полного дерева с корнем в вершине a[0]. В обобщенной двоичной куче верхушка дерева срезана. Поэтому в нем допускается более одной корневой вершины.

5.3. Верификация программ с неоднозначной спецификацией

Спецификация подпрограмм heapify и siftDown неоднозначна, поскольку по исходному массиву можно построить много разных куч, перестановочных с исходным массивом. Проблема состоит в том, что спецификация должна быть достаточно сильной, чтобы доказать корректность программы. Для однозначных спецификаций такой проблемы нет, поскольку спецификация тождественна программе.

Неоднозначность спецификации принципиально осложняет спецификацию и верификацию. Определить требуемое усиление спецификации априори проблематично. Неудачное усиление может сильно осложнить верификацию. Так, в предыдущем релизе верификации программы sort в 2019г. были использованы усиления, сделанные в более

ранней верификации [12]. В текущем релизе было решено ограничиться лишь очевидными условиями, а необходимые усиления аккуратно определить в процессе доказательства и попытаться сделать их минимальными. Далее отмечается, какие части спецификации были внесены позже. В частности, в конце процесса верификации обнаружилось, что для постусловия программы siftDown необходимо использовать предикат permE вместо perm.

Несмотря на принятую стратегию аккуратного усиления спецификации, верификация оказалась сложной и трудоемкой.

5.4. Спецификация программы heapify

В программе heapify из соображений удобства заменим имена переменных. Определим спецификацию программы heapify.

```
heapify(Ar a, int i: Ar b)
   pre -1<=i<num & heap(a, i + 1, num)
   post perm(a, b) & heap(b)
   measure i
{      if (i < 0) b = a
       else { SiftDown(a, i, num: Ar c);  heapify(c, i - 1: b) }
}
```

Программа heapify по массиву a строит двоичную кучу b. Создание кучи начинается с конца массива a. В соответствии с начальным вызовом heapify(base, num/2 - 1: Ar base1) в программе sort куча считается построенной для диапазона num/2 - 2, поскольку состоит из одиночных вершин. На очередном шаге работы программы heapify(a, i: b) имеется обобщенная куча в диапазоне от i+1 до num-1. Необходимо построить обобщенную кучу от i до num-1. Для этого следует некоторым способом переместить элемент a[i] внутрь дерева с корнем в вершине с индексом i. Эта операция называется *просеиванием вниз* и реализуется программой SiftDown.

Ограничение -1<=i<num в предусловии появляется в результате дедуктивной верификации.

Отметим, что вместо условия i < 0 было бы лучше использовать i <= 0. Но это будет другая программа.

5.5. Спецификация программы sorting

Определим программу sorting вместе со спецификацией. Предварительно изменим имена переменных.

```
sorting(Ar b, int i: Ar b')
   pre  -1<=i<num & heap(b, 0, i+1) & sorted(b, i+1) & twoPart(b, i+1)
    post sorted(b') & perm(b, b')
    measure i
{       if (i <= 0) b' = b
        else {  swap(b, 0, i: Ar b1);
                siftDown(b1, 0, i: Ar c);
                sorting(c, i - 1: b')
        }
}
```

Программа sorting реализует сортировку массива b, состоящего из двух частей. Правая часть состоит из элементов b[i+1], b[i+2], …, b[num-1] и уже отсортирована. Левая часть в диапазоне от 0 до i является двоичной кучей. Для начального вызова sorting(base1, num - 1: base') в программе sort левая часть является пустой.

Элемент b[0] является максимальным элементом массива. В отсортированном массиве b' он должен находиться в позиции i. Обменяем местами нулевой элемент и элемент i. После обмена элементов массив b перестал быть кучей. Свойство кучи можно восстановить, если запустить программу siftDown для нулевого элемента.

Условие -1<=i<num в предусловии было вставлено в процессе верификации. Условие twoPart(b, i + 1) было вставлено при верификации ранее еще в работе [12].

```
    formula twoPart(Ar b, nat m) = m < num ⇒ b[0] <= b[m];
```

Данное условие необходимо для доказательства сортированности и отражает тот факт, что левая часть массива b должна быть не больше правой части.

5.6. Спецификация программы siftDown

Представим программу siftDown.

```
siftDown(Ar a, int k, r, m: Ar b)
  pre psiftD(a, k, r, m)
  post qsiftD(a, b, k, r, m)
  measure (r>= m)? 0 : m - r
{     int c = r *2 + 1;
      if (c >= m) b = a
      else { if (c < m - 1 && cmp(a[c], a[c+1]) < 0) c1 = c+1 else c1 = c;
             if (cmp(a[r], a[c1]) >= 0) b = a
             else { swap(a, r, c1: Ar a1); siftDown(a1, c1, m: b)};
      }
}
```

Предусловие psiftD и постусловие qsiftD будут определены ниже.

В предположении, что обобщенная куча построена от r+1 до m-1 (m ≤ num), программа siftDown строит обобщенную кучу от r до m-1. Элемент a[r] сравнивается с наибольшим из потомков и обменивается с ним операцией swap, если потомок оказался большим. После обмена элементов продолжается просеивание вниз исходного элемента a[r], находящегося в позиции потомка.

Спецификация siftDown очевидно должна содержать heap(a, r+1, m) в составе предусловия и perm(a, b) & heap(b, r, m) в постусловии. После нескольких этапов уточнений спецификации были получены следующие достаточно сложные предусловие и постусловие. Рассмотрим предусловие:

```
formula psiftD(Ar a, nat k, r, m) = k<=r<m<=num & heapH(a, k, r, m);
```

Условия heap(a, r+1, m) оказалось недостаточно, когда позиция r находится в глубине дерева и не является корневой. Предикат heapH определяет условие, что массив a на отрезке от k до m-1 обладает свойством кучи за возможным исключением позиции «дыры» r, где реализуется особое условие. Здесь также пришлось ввести дополнительный параметр k.

```
formula heapH(Ar a, nat k, r, m) =
        (k <= father(r) ⇒ heapR(a, r, m)) &
        (∀ nat j.  k <= j < m & j ≠ r  ⇒ heapJ(a, j, m))
```

В позиции «дыры» r свойство кучи реализуется для отца вершины r и любого из потомков вершины r.

```
formula heapR(Ar a, nat r, m) = ∀nat j=k..m-1. father(j) = r ⇒ cmp(a[father r], a[j]) >=0;
```

Рассмотрим постусловие:

```
formula qsiftD(Ar a, b, nat k, r, m) = permE(a, b, r, m) &
     (∀nat j. j<num ⇒ if inTree(r, m, j)  then  heapJ(b, j, m)  else  b[j] = a[j]) &
     ( k<=father(r) ⇒ cmp(a[father(r)], b[r]) >=0 )
```

Вместо perm(a, b) потребовалось более точное условие permE(a, b, r, m).

Недостаточно точным оказалось исходное установленное условие heap(b, r, m). Во втором конъюнкте формулы qsiftD уточняется, что свойство кучи реализуется только в дереве с корнем r. А за пределами этого дерева итоговый массив b должен совпадать с исходным массивом a.

Потребовалось дополнительное условие, указанное в третьем конъюнкте формулы qsiftD. Свойство кучи должно выполняться для отца вершины r и самой вершиной r.

Предикат inTree определяет принадлежность вершины j дереву с корнем r:

```
formula inTree(nat r, m, j) = r<=j<m & path(r, j);
```

Дерево с корнем r должно находиться в пределах обобщенной двоичной кучи от r до m-1.

Предикат path(r, j) определяет существование пути от вершины r к вершине j в дереве с корнем r. Очевидное рекурсивное определение предиката path оказалось недопустимым в языке спецификаций why3. Поэтому было использовано следующее индуктивное определение:

```
inductive path(int n, p) =
    | Ref:  ∀ int n. n>=0 ⇒ path(n, n)
    | Lep:  ∀ int n, p. path(n, p) ⇒ path(n, left (p))
    | Rip:  ∀ int n, p. path(n, p) ⇒ path(n, right(p))
```

Отметим, что вместо cmp(a[c], a[c+1]) < 0 можно было бы написать cmp(a[c], a[c+1])<=0, однако тогда после трансформаций программа не совпадет с исходной программой на языке Си в Приложении 1.

6. Процесс дедуктивной верификации программы sort

Для предикатной программы пирамидальной сортировки, состоящей из программ sort, heapify, sorting и siftDown, построены формулы корректности по правилам, описанным в [11]. Процесс построения формул корректности детально документирован в Приложении 2. В Приложения 3 определена теория на языке P с набором формул корректности. Далее эта теория была закодирована на языке спецификаций why3 системы Why3 [24].

Использовалась система Why3 версии 1.1.1 с SMT-решателями CVC3 версии 2.4.1, CVC4 версии 1.6, Z3 версии 4.7.1 и Gappa версии 1.3.2. Впрочем, SMT-решатель Gappa оказался бесполезным. Стандартное время запуска SMT-решателей: в начале процесса доказательства – 22 сек, в конце – 47 сек. SMT-решатели запускались в параллельном режиме на трех 64-разрядных процессорах 3.3 GHz.

Исходное задание содержало 16 формул корректности. В процессе доказательства введено 95 лемм. Большинство формул корректности и лемм доказано с помощью SMT-решателей при использовании трансформаций. В системе Coq [17] проведено 35 различных доказательств формул корректности, лемм и их частей. Для сложных формул корректности выстраивалась цепочка конкретизирующих лемм. Это помогло лишь частично. Процесс доказательства в целом оказался заметно сложнее, чем в предыдущей версии в 2019г..

В процессе доказательства обнаружена недоказуемость формул корректности. Проведено уточнение предусловий в программах heapify и sorting. Исправлены мелкие ошибки. Ошибочная перестановка параметров формулы heapJ была обнаружена достаточно поздно.

Учитывая неудачный опыт предыдущих попыток [12], почти все предыдущие уточнения спецификаций были отвергнуты. В текущем релизе сначала используются лишь очевидные

условия, а необходимые усиления аккуратно определяются в процессе доказательства с целью ограничиться минимальными усилениями. С этой целью пришлось достаточно глубоко пройти в процессе доказательства для определения нужных уточнений.

Проведено три этапа уточнения спецификаций. Сначала определена необходимость предиката heapH, определяющего кучу с «дырой». Далее установлено, что необходимо условие того, что изменения реализуются лишь в пределах дерева (второй конъюнкт qsiftD). Определен предикат inTree , а затем – предикат path. В конце введено свойство кучи для отца текущей вершины r. Это уточнение намного проще введенного в релизе 2019г. предиката root.

Несмотря на все принятые меры, процесс доказательства оказался сложным т трудоемким. SMT-решатели достаточно часто не справлялись с вроде бы простыми формулами. Приходилось переводить доказательство в систему Coq, где доказательство редко было простым. В конце процесса верификации SMT-решатели перестали доказывать пару формул, которые ранее быстро доказывали.

Итоги верификации. Все формулы корректности и все дополнительные леммы полностью доказаны. Верификация проводилась в течение более двух недель параллельно с изучением и освоением системы интерактивного доказательства Coq [17].

7. Обзор работ

В нашей работе [12] проводилась дедуктивная верификация трех алгоритмов пирамидальной сортировки: классического алгоритма Дж. Вильямса [25], алгоритма Флойда [20] и улучшенного алгоритма [23] – самого быстрого алгоритма сортировки в то время. Для классического алгоритма верификация не была доведена до конца. Основной целью была верификация самого быстрого алгоритма. Материал работы [12] был заимствован в релизе 2019г., однако это оказало скорее негативное влияние. Алгоритм и спецификации пришлось сводить от массивов a[1..n] к массивам вида a[0..n-1], алгоритм был существенно модифицирован под исходную программу в Приложении 1. Описание программы было полностью заменено. Заимствованные леммы оказались бесполезными.

Наиболее значимой является работа [19] по дедуктивной верификации трех алгоритмов сортировки: insertion sort, quicksort и heapsort в системе Coq [17]. Файлы, иллюстрирующие процесс доказательства корректности heapsort приведены на странице:

http://why.lri.fr/examples/index.en.html.

Программа и спецификации писались на языке WhyML [24], а доказательство проводилось в Coq. Спецификация программы downheap существенно проще аналогичной siftDown. Не используется свойство кучи с дырой. Нет требования сохранения равенства элементов вне сортируемого поддерева. Вместо этого используется предикат inftree, декларирующий сохранение после работы downheap следующего свойства: для произвольного v все элементы поддерева меньше данного v. Сопоставление по объему и сложности доказательства провести трудно.

В работе [18] на примере программы heapsort демонстрируется технология программирования на базе инвариантов с использованием системы PVS [21]. В спецификации siftdown используется свойство кучи с дырой, но нет предиката, аналогичного inTree или inftree. Поэтому есть серьезные сомнения, что предложенные спецификации достаточны для доказательства корректности heapsort.

Дедуктивная верификация программы heapsort проводилась в работе [5] с использованием среды верификации Isabelle/HOL. В спецификации siftDown свойство кучи с дырой не используется. В постусловии используется предикат равенства элементов до и после на диапазоне. Как показал наш опыт, этого недостаточно. Есть сомнения, что дедуктивная верификация была доведена до конца.

8. Заключение

Проведена обратная трансформация программы пирамидальной сортировки sort.c из библиотеки ядра ОС Linux. Использовались трансформация запроцедуривания и трансформация, обратная уменьшению силы операций. Эти трансформации ранее не использовались. Для полученной предикатной программы написаны спецификации. По программе и спецификациям построены формулы корректности, оформленных в виде теории, перенесенной на язык спецификаций why3. Доказательство проводилось в системах Why3[24] и Coq [17]. В процессе верификации уточнялись спецификации. Доказательство формул корректности и всех лемм проведено полностью. Процесс дедуктивной верификации оказался сложным и трудоемким.

Итоги верификации. Библиотечная программа sort корректна относительно спецификации. Для функции cmp_func определена спецификация в виде следующей теории.

```
theory Compare {
type CMP= predicate(T, T: int)
    CMP cmp;
    axiom Refl :   ∀ x: T. cmp(x, x) = 0
    axiom Simm :   ∀ T x, y. cmp(x, y) = - cmp(y, x);
    axiom TotalLe : ∀ T x, y. cmp(x, y) <=0 or cmp(y, x) <=0;
    axiom TransLe : ∀ T x, y, z. cmp(x, y)<=0 & cmp(y, z)<=0 ⇒ cmp(x,z)<=0;
}
```

Аксиома антисимметричности: x <= y & y <= x ⇒ x = y – не использовалась.

Следует отметить, что спецификация и верификации исходной программы sort.c существующими инструментами была бы на порядок сложнее проделанной и описанной в настоящей работе.

Доступна полная версия текста настоящей работы: https://persons.iis.nsk.su/files/persons/pages/sort9.pdf.

Список литературы

1. Доказательство правил корректности операторов предикатной программы. [Электронный ресурс]. URL:http://www.iis.nsk.su/persons/vshel/files/rules.zip (дата обращения 02.09.2020)
2. Ефремов Д.В, Мандрыкин М.У. Формальная верификация библиотечных функций ядра Linux // Труды ИСП РАН, том 29, вып. 6, 2017. С. 49-76. DOI: 10.15514/ISPRAS-2017-29(6)-3
3. Каблуков И.В., Шелехов В.И. Реализация оптимизирующих трансформаций в системе предикатного программирования // Системная информатика, № 11. Новосибирск, 2017. С. 21-48. Электрон. журн. 2018. http://persons.iis.nsk.su/files/persons/pages/opttransform4.pdf
4. Карнаухов Н.С., Першин Д.Ю., Шелехов В.И. Язык предикатного программирования Р // Новосибирск, 2018. 42с. [Электронный ресурс]. URL: http://persons.iis.nsk.su/files/persons/pages/plang14.pdf (дата обращения 02.09.2020)
5. Ковалев М.С., Далингер Я.М., Мяготин А.В. Формальная верификация программной реализации алгоритма пирамидальной сортировки на языке Си-0 // Научно-технические ведомости СПбГПУ. 2010. № 4.С.83-92.
6. Чушкин М.С. Система дедуктивной верификации предикатных программ // «Программная инженерия». 2016. № 5. С. 202-210. [Электронный ресурс]. URL: http://persons.iis.nsk.su/files/persons/pages/paper.pdf. (дата обращения 02.09.2020)
7. Шелехов В.И. Верификация и синтез эффективных программ стандартных функций в технологии предикатного программирования // Программная инженерия, 2011, № 2. С. 14-21. [Электронный ресурс]. URL: https://www.iis.nsk.su/files/preprints/154.pdf (дата обращения 02.09.2020)
8. Шелехов В.И. Дедуктивная верификация и оптимизация предикатной программы конкатенации строк // Системная информатика, № 12. Новосибирск, 2018. С. 61-84. http://persons.iis.nsk.su/files/persons/pages/strcat.pdf

9. Шелехов В.И. Дедуктивная верификация программы конкатенации строк с применением обратной трансформации // Знания-Онтологии-Теории (ЗОНТ-19). Новосибирск, 2019. 19с. [Электронный ресурс]. URL: http://persons.iis.nsk.su/files/persons/pages/logcflc1.pdf (дата обращения 02.09.2020)
10. Шелехов В.И. Классификация программ, ориентированная на технологию программирования // «Программная инженерия», Том 7, № 12, 2016. С. 531–538. [Электронный ресурс]. URL: http://persons.iis.nsk.su/files/persons/pages/prog.pdf . (дата обращения 02.09.2020)
11. Шелехов В.И. Правила доказательства корректности предикатных программ // Новосибирск, ИСИ СО РАН, 2019. [Электронный ресурс]. URL: http://persons.iis.nsk.su/files/persons/pages/prrules.pdf (дата обращения 02.09.2020)
12. Шелехов В.И. Разработка и верификация алгоритмов пирамидальной сортировки в технологии предикатного программирования // Новосибирск, 2012. 30с. (Препр. / ИСИ СО РАН. № 164). [Электронный ресурс]. URL: https://www.iis.nsk.su/files/preprints/164.pdf (дата обращения 02.09.2020)
13. Шелехов В.И. Семантика языка предикатного программирования // ЗОНТ-15. Новосибирск, 2015. 13с.http://persons.iis.nsk.su/files/persons/pages/semZont1.pdf . (дата обращения 02.09.2020)
14. Шелехов В.И. Синтез операторов предикатной программы // Труды конф. «Языки программирования и компиляторы ’2017», Ростов-на-Дону. 2017. С.258-262. [Электронный ресурс]. URL: http://persons.iis.nsk.su/files/persons/pages/sintr.pdf (дата обращения 12.11.2018).
15. Шелехов В.И., Чушкин М.С. Верификация программы быстрой сортировки с двумя опорными элементами // Научный сервис в сети Интернет. М.: ИПМ им. М.В.Келдыша, 2018. 26с. [Электронный ресурс]. URL: http://persons.iis.nsk.su/files/persons/pages/dqsort.pdf (дата обращения 12.11.2018).
16. AstraVer Toolset: инструменты дедуктивной верификации моделей и механизмов защиты ОС // ИСП РАН. URL: http://linuxtesting.org/astraver, 15.10.2017 (дата обращения 30.11.2018).
17. The Coq Proof Assistant. [Электронный ресурс]. URL: http://coq.inria.fr (дата обращения 02.09.2020)
18. Eriksson J. and Back R.-J. Applying PVS Background Theories and Proof Strategies in Invariant Based Programming // LNCS 6447, 2010. P. 24–39.
19. Filliâtre J.-C., Magaud N. Certification of sorting algorithms in the system Coq // Theorem Proving in Higher Order Logics: Emerging Trends, 1999.
20. Floyd R.W. Algorithm 245 – Treesort **3** // Commun. ACM. 1964. Vol. 7 (12). P. 701.
21. PVS Specification and Verification System. SRI International. [Электронный ресурс]. URL: http://pvs.csl.sri.com/ . (дата обращения 02.09.2020)
22. Shelekhov V. I. Verification and Synthesis of Addition Programs under the Rules of Correctness of Statements // Automatic Control and Computer Sciences. 2011. Vol. 45, No. 7, P. 421–427.

23. Wang X.D., Wu Y. J. An improved HEAPSORT algorithm with n logn – 0.788928n comparisons in the worst case // J. Computer Science and Technology. 2007. Vol. 22 (6). P. 898–903.
24. Why 3. Where Programs Meet Provers. [Электронный ресурс]. URL: http://why3.lri.fr (дата обращения 02.09.2020)
25. Williams J.W.J., Algorithm 232 // Commun. ACM. 1964. P. 347–348.

Приложение 1

Исходная программа sort на языке Си

```
#define KERNRELEASE "TEST"
#define CONFIG_HAVE_EFFICIENT_UNALIGNED_ACCESS 1
#define __take_second_arg(__ignored,val,...) val
#define ____is_defined(arg1_or_junk) __take_second_arg(arg1_or_junk 1, 0)
#define ____or(arg1_or_junk,y) __take_second_arg(arg1_or_junk 1, y)
#define ___is_defined(val) ____is_defined(__ARG_PLACEHOLDER_ ##val)
#define ___or(x,y) ____or(__ARG_PLACEHOLDER_ ##x, y)
#define __is_defined(x) ___is_defined(x)
#define __or(x,y) ___or(x, y)
#define IS_BUILTIN(option) __is_defined(option)
#define IS_MODULE(option) __is_defined(option ##_MODULE)
#define IS_ENABLED(option) __or(IS_BUILTIN(option), IS_MODULE(option))
typedef unsigned long __kernel_ulong_t;
typedef unsigned int u32;
typedef unsigned long long u64;
typedef __kernel_ulong_t __kernel_size_t;
typedef __kernel_size_t          size_t;

//-----------------------------------------------------------------------------

static void generic_swap(void *a, void *b, int size)
{
    char t;

    do {
            t = *(char *)a;
            *(char *)a++ = *(char *)b;
            *(char *)b++ = t;
    } while (--size > 0);
}

static void u32_swap(void *a, void *b, int size)
{
    u32 t = *(u32 *)a;
    *(u32 *)a = *(u32 *)b;
    *(u32 *)b = t;
}

static void u64_swap(void *a, void *b, int size)
{
    u64 t = *(u64 *)a;
    *(u64 *)a = *(u64 *)b;
    *(u64 *)b = t;
}

static int alignment_ok(const void *base, int align)
```

```
{
    return IS_ENABLED(CONFIG_HAVE_EFFICIENT_UNALIGNED_ACCESS) ||
            ((unsigned long)base & (align - 1)) == 0;
}

void sort(void *base, size_t num, size_t size,
      int (*cmp_func)(const void *, const void *),
      void (*swap_func)(void *, void *, int size))
{
    /* pre-scale counters for performance */
    int i = (num/2 - 1) * size, n = num * size, c, r;

    if (!swap_func) {
            if (size == 4 && alignment_ok(base, 4))
                    swap_func = u32_swap;
            else if (size == 8 && alignment_ok(base, 8))
                    swap_func = u64_swap;
            else
                    swap_func = generic_swap;
    }

    /* heapify */
    for ( ; i >= 0; i -= size) {
            for (r = i; r * 2 + size < n; r  = c) {
                    c = r * 2 + size;
                    if (c < n - size &&
                                    cmp_func(base + c, base + c + size) < 0)
                            c += size;
                    if (cmp_func(base + r, base + c) >= 0)
                            break;
                    swap_func(base + r, base + c, size);
            }
    }

    /* sort */
    for (i = n - size; i > 0; i -= size) {
            swap_func(base, base + i, size);
            for (r = 0; r * 2 + size < i; r = c) {
                    c = r * 2 + size;
                    if (c < i - size &&
                                    cmp_func(base + c, base + c + size) < 0)
                            c += size;
                    if (cmp_func(base + r, base + c) >= 0)
                            break;
                    swap_func(base + r, base + c, size);
            }
    }
}
```

Приложение 3

Теории для доказательства формул корректности

Соберем все сгенерированные формулы корректности в теорию.

```
   theory Sort {
nat num;
type T;
nat Di = 0..num-1;
type Ar = array (T, Di);
cmp(T, T: int);
swap(Ar a, Di j, k: Ar a') post exchange(a, a', j, k);
Ar base;
     axiom Refl :   ∀ x: T. cmp(x, x) = 0
     axiom Simm :   ∀ T x, y. cmp(x, y) = - cmp(y, x);
     axiom TotalLe : ∀ T x, y. cmp(x, y) <=0 or cmp(y, x) <=0;
     axiom TransLe : ∀ T x, y, z. cmp(x, y)<=0 & cmp(y, z)<=0 ⇒ cmp(x,z)<=0;
formula sorted(Ar a, nat m) =  ∀ k, j= m..num-1. k < j ⇒  cmp(a[k], a[j]) <= 0;

formula perm(Ar a, b) = permut_all(a, b);
formula permE(Ar a, b, nat m, n) = permut_sub(a, b, r, m);

formula left(nat j : nat) = j * 2 + 1;
formula right(nat j : nat) = j * 2 + 2;
formula father(nat j : int) = (j-1) / 2;
   formula heap(Ar a) = ∀ nat j = 0 .. num-1. heapJ(a, num, j)
   formula heapJ(Ar a, nat j, m) =    (left(j) < m ⇒ cmp(a[j], a[left(j)])>=0)) &
                                      (right(j) < m ⇒ cmp(a[j], a[right(j)])) .
   formula heap(Ar a, nat i, m) = ∀ nat j = i .. m-1. heapJ(a, j, m);
formula twoPart(Ar b, nat m) = m < num ⇒ b[0] <= b[m];
   formula heapR(Ar a, nat r, m) = ∀nat j=k..m-1. father(j) = r ⇒ cmp(a[father r], a[j])
   >=0;
   formula heapH(Ar a, nat k, r, m) =
              (k <= father(r) ⇒ heapR(a, r, m)) &
              (∀ nat j.  k <= j < m & j ≠ r  ⇒ heapJ(a, j, m))
   formula psiftD(Ar a, nat k, r, m) = k<=r<m<=num & heapH(a, k, r, m);
   formula inTree(nat r, m, j) = r<=j<m & path(r, j);
   inductive path(int n, p) =
        | Ref:   ∀ int n. n>=0 ⇒ path(n, n)
        | Lep:   ∀ int n, p. path(n, p) ⇒ path(n, left (p))
        | Rip:   ∀ int n, p. path(n, p) ⇒ path(n, right(p))
   formula qsiftD(Ar a, b, nat k, r, m) = permE(a, b, r, m) &
          (∀nat j. j<num ⇒ if inTree(r, m, j)  then  heapJ(b, j, m)  else  b[j] = a[j]) &
          ( k<=father(r) ⇒ cmp(a[father(r)], b[r]) >=0 )
formula qsort(Ar base, base9) = sorted(base9, 0) & perm(base, base9);
formula pheapify(Ar a, int i) = -1<=i<num & heap(a, i+1, num);
formula qheapify(Ar a, b) = perm(a, b) & heap(b, 0, num);
```

formula psorting(Ar b, **int** i) = -1<=i<num & heap(b, 0, i+1) & sorted(b, i+1) & twoPart(b, i+1);
formula qsorting(Ar b, b9) = sorted(b9) & perm(b, b9);

RS1: pheapify(base, (num **div** 2) – 1);
RS2: qheapify(base, b) ⇒ psorting(b, num-1);
RS3: qheapify(base, b) & qsorting(b, base9) ⇒ qsort(base, base9);

COR1: pheapify(a, i) & i < 0 & b = a ⇒ qheapify(a, b);
RS4: pheapify(a, i) & i >= 0 ⇒ psiftD(a, i, num);
RS5: pheapify(a, i) & i >= 0 & qsiftD(a, c, i, num) ⇒ pheapify(c, i - 1) & i-1<i;
RS6: pheapify(a, i) & i >= 0 & qsiftD(a, c, i, num) & qheapify(c, b) ⇒ qheapify(a, b);
COR2: psorting(b, i) & i <= 0 & b9 = b ⇒ qsorting(b, b9);
QS1: psorting(b, i) & i > 0 ⇒ 0<num & i<num;
RS7: psorting(b, i) & i > 0 & exchange(b, b1, 0, i) ⇒ psiftD(b1, 0, i);
RS8: psorting(b, i) & i > 0 & exchange(b, b1, 0, i) & qsiftD(b1, c, 0, i) ⇒
psorting(c, i - 1) & i-1<i;
RS9: psorting(b, i) & i > 0 & exchange(b, b1, 0, i) & qsiftD(b1, c, 0, i) &
qsorting(c, b9) ⇒ qsorting(b, b9)
} Sort

theory SiftDown {
import Sort;
COR3: psiftD(a, r, m) & c = r *2 + 1 & c >= m & b = a ⇒ qsiftD(a, b, r, m);
formula cc2(Ar a, **nat** m, c, c1) =
if (c < m - 1 && cmp(a[c], a[c+1]) < 0) c1 = c+1 **else** c1 = c;
formula fcc(Ar a, **nat** r, m, c, c1) =
psiftD(a, r, m) & c = r *2 + 1 & c < m & cc2(a, m, c, c1);
COR4: fcc(a, r, m, c, c1) & (cmp(a[r], a[c1]) >= 0) & b = a ⇒ qsiftD(a, b, r, m);
formula h(**nat** r, m**: nat**) = (r>= m)? 0 **:** m - r
RB1: fcc(a, r, m, c, c1) & cmp(a[r], a[c1]) < 0 & exchange(a, a1, r, c1) ⇒
0<= c1 < num & psiftD(a1, c1, m) & h(c1, m)<h(r, m);
RB2: fcc(a, r, m, c, c1) & cmp(a[r], a[c1]) < 0 & exchange(a, a1, r, c1) & qsiftD(a1, b, c1, m) ⇒
qsiftD(a, b, r, m);
} SiftDown

УДК 004.05

Трансформация, спецификация и верификация программы вычисления числа элементов множества, представленного в виде битовой шкалы

Тумуров Э.Г. (Институт систем информатики СО РАН),
Шелехов В.И. (Институт систем информатики СО РАН, Новосибирский государственный университет)

Описывается трансформация программы memweight из библиотеки ОС Linux, устраняющая указатели. Далее программа трансформируется на язык предикатного программирования P. Для предикатной программы, полученной в результате серии упрощающих трансформаций, строится спецификация и проводится дедуктивная верификация. Для упрощения верификации в рамках спецификации строится модель внутреннего состояния исполняемой программы. Верификация программы memweight реализована в системе автоматического доказательства Why3.

Ключевые слова: *дедуктивная верификация, трансформации программ, автоматическое доказательство, функциональное программирование, предикатное программирование.*

1. Введение

В библиотеке ядра операционной системы (ОС) Linux имеется программа memweight на языке Си, вычисляющая число элементов множества, представленного в виде битовой шкалы. Код программы memweight приводится в Приложении 1. Программа более известна как программа вычисления *веса* Хэмминга. Она вычисляет число единиц в битовом представлении памяти. Параметрами программы memweight являются: ptr – указатель на байтовый массив, bytes – число байтов в массиве. В указанном массиве памяти нужно определить число единичных битов.

Функция memweight многократно вызывается в программе ядра ОС Linux. Поэтому высоки требования к ее эффективности. Используется сложный эффективный алгоритм, так называемая битовая магия [4] для 32- или 64-битных слов.

Дедуктивная верификация намного проще и быстрее для функциональных программ [12], чем для аналогичных императивных программ, потому что функциональная программа проще эквивалентной императивной программы. Этот факт отмечается разными исследователями. Преимущество по времени верификации для предикатных программ [3] оценивалось примерно в 5 раз. Для программы быстрой сортировки с двумя опорными элементами зафиксировано преимущество в 10 раз [11]. Отметим высокую сложность проведения формальной спецификации и дедуктивной верификации императивных программ.

Причина сложности императивных программ в том, что указатели, конструкции необходимые для оптимизации программ, существенно усложняют логику императивных программ. Для упрощения императивных программ применяются трансформации, устраняющие указатели в императивной программе [8]. Операции с указателями заменяются эквивалентными действиями без указателей. Далее к полученной программе применяются трансформации, превращающие ее в эквивалентную предикатную программу. Здесь проводится оформление аргументов и результатов функций и замена каждого цикла соответствующей рекурсивной программой.

Для предикатной программы memweight, полученной в результате трансформации, конструируются формальные спецификации в виде предусловия и постусловия для каждой функции, входящей в состав предикатной программы. Для упрощения верификации в рамках спецификации строится модель внутреннего состояния исполняемой программы. Модель экстрагируется в виде независимой теории из константной части программы. Данный метод позволяет успешно декомпозировать сложные доказательства.

Далее строятся формулы корректности предикатной программы memweight относительно спецификации применением системы правил [11]. Доказана корректность правил [1]. Совокупность формул корректности вместе с описаниями типов и переменных оформляется в виде набора теорий. Эти теории транслируются на язык спецификаций why3 [19]. Далее в системе дедуктивной верификации Why3 [19] реализуется процесс доказательства формул корректности.

В версии 1.3.1 системы Why3 в дополнении к возможности использования множества эффективных SMT-решателей реализована полноценная система интерактивного доказательства. Благодаря этому появилась возможность полностью реализовать доказательство в рамках системы Why3 без выхода в систему Coq [17], как это было ранее [6, 7]. Дополнительным эффективным методом является использование аппарата лемма-

функций [18] для доказательства сложной леммы, в частности, требующей доказательства по индукции. Здесь строится вспомогательная предикатная программа, спецификация которой совпадает с исходной леммой.

Во втором разделе настоящей работы дается краткое описание языка предикатного программирования. В третьем разделе описывается трансформация исходной программы memweight с получением эквивалентной предикатной программы. В следующем разделе описывается процесс спецификации предикатной программы. Особенности процесса дедуктивной верификации предикатной программы в системе Why3[19] описывается в шестом разделе. Исправлено значительное число ошибок спецификации. Далее обзор работ. В заключении суммируются итоги верификации. В Приложении 1 код исходной программы memweight. В Приложении 2 представлены полные теории, созданные для доказательства формул корректности на языке Why3.

2. Язык предикатного программирования

Полная предикатная программа состоит из набора рекурсивных *предикатных программ* на языке P [3] следующего вида:

```
<имя программы>(<описания аргументов>: <описания результатов>)
pre <предусловие>
post <постусловие>
measure <выражение>
{ <оператор> }
```

Предусловие и постусловие являются формулами на языке исчисления предикатов. Они обязательны при дедуктивной верификации [5, 9, 10]. Мера задается только для рекурсивных программ и используется для верификации.

Ниже представлены основные конструкции языка P: оператор присваивания, блок (оператор суперпозиции), параллельный оператор, условный оператор, вызов программы и описание переменных, используемое для аргументов, результатов и локальных переменных.

```
<переменная> = <выражение>
{<оператор1>; <оператор2>}
<оператор1> || <оператор2>
if (<логическое выражение>) <оператор1> else <оператор2>
<имя программы>(<список аргументов>: <список результатов>)
<тип> <пробел> <список имен переменных>
```

Всякая переменная характеризуется *типом* – множеством допустимых значений. *Описание типа* **type** T(p) = D с возможными параметрами p связывает имя типа T с его

изображением D. Типы **bool**, **int**, **real** и **char** являются *примитивными*. Значением типа **array**(T_e, T_i) является *массив* с *элементами массива* типа T_e и *индексами* конечного типа T_i. Тип массива является предикатным типом, его значения (массивы) являются тотальными и однозначными предикатами.

Пусть E(x) – логическое выражение. Тип **subtype**(T x**:** E(x)) определяет *подтип* типа T при истинном предикате E(x), т.е. множество {x∈T | E(x)}. Определенный в языке P тип целых чисел **nat** представляется описанием:

type nat = **subtype**(**int** x: x ≥ 0) .

Допускаются подтипы, параметризуемые переменными. Примером является тип *диапазона* целых чисел:

type Diap(**nat** n) = **subtype**(**int** x: x ≥ 1 & x ≤ n) .

В языке P для изображения типа диапазона используется конструкция 1..n.

Описания типов переменных являются частью спецификации программы. Описание переменной T x есть утверждение x∈T, которое становится частью предусловия, если x – аргумент предикатной программы, или частью постусловия, если x – результат программы. При этом утверждение x∈T обычно не пишется в составе предусловия или постусловия, хотя предполагается.

В языке предикатного программирования P [3] нет указателей, серьезно усложняющих программу. Вместо указателей используются объекты алгебраических типов: списки и деревья. Предикатная программа существенно проще в сравнении с императивной программой, реализующей тот же алгоритм. Эффективность предикатных программ достигается применением *оптимизирующих трансформаций* [2]. Они определяют отличную от классической оптимизацию среднего уровня с переводом предикатной программы в эффективную императивную программу.

Базовыми трансформациями являются:

- склеивание переменных, реализующее замену нескольких переменных одной;
- замена хвостовой рекурсии циклом;
- открытая подстановка программы на место ее вызова;
- кодирование объектов алгебраических типов (списков и деревьев) при помощи массивов и указателей.

3. Трансформации программы вычисления числа элементов множества

Трансформации направлены на упрощение исходной императивной программы. В итоге получается эквивалентная предикатная программа. Сначала проводится нормализация программы. Удаляются конструкции, ориентированные на трансляцию. Раскрываются умолчания для условий в условных операторах и заголовках циклов. Далее устраняются указатели. Вместо них в программе появляются явные обращения к элементам массивов. Наконец, циклы преобразуются в рекурсивные программы.

3.1. Описание программы memweight

Представим заголовок программы:

```
size_t memweight(const void *ptr, size_t bytes)
```

Дана область памяти, начало которой фиксируется указателем ptr. Размером области в байтах определяется вторым параметром bytes. Программа вычисляет ***вес*** (вес Хэмминга) области памяти – количество единичных битов в двоичном представлении области памяти. Программа более известна как программа вычисления веса Хэмминга.

Полный код программы memweight вместе со всеми вызываемыми функциями в исходном ненормализованном виде приведен в Приложении 1. Далее представлена часть программы после проведения этапа нормализации, на котором устраняются макросы, раскрываются умолчания для выражений нелогического типа в логических позициях и проводятся другие преобразования.

Ниже приводится код главной программы memweight. Этот код и его описание доступны также по ссылке https://elixir.bootlin.com/linux/v5.7.8/source/lib/memweight.c .

```
const INT_MAX = 0x7fffffff;
const BITS_PER_LONG = 64;

size_t memweight(const void *ptr, size_t bytes)
{
        size_t ret = 0;
        size_t longs;
        const unsigned char *bitmap = ptr;
        for (; bytes > 0 && ((unsigned long)bitmap) % sizeof(long) != 0;
                        bytes--, bitmap++)
                ret += hweight8(*bitmap);
        longs = bytes / sizeof(long);
        if (longs != 0) {
                assert(longs < INT_MAX / BITS_PER_LONG);
                ret += bitmap_weight((unsigned long *)bitmap,
                                longs * BITS_PER_LONG);
                bytes -= longs * sizeof(long);
                bitmap += longs * sizeof(long);
        }
        for (; bytes > 0; bytes--, bitmap++)
                ret += hweight8(*bitmap);
        return ret;
}
```

В теле главной программы memweight используются две функции: hweight8 для подсчета числа единиц в байте и bitmap_weight для подсчета единиц в массиве слов размером 32 или 64 бита в зависимости от архитектуры. Функция bitmap_weight значительно более эффективна в сравнении с побайтовой обработкой через hweight8.

Пословная адресация памяти в ряде архитектур ЭВМ возможна лишь с границы слова, т.е. когда адрес начала памяти кратен 4 для 32-битных слов (8 для 64-битных слов). Поэтому функция bitmap_weight может быть запущена лишь с границы слова. С учетом этого исходная память, заданная параметрами memweight, разделяется на три части: короткий байтовый массив до первой границы слова, словный массив и короткий байтовый массив за последним словом (см. Рис.1 и 2). Словный массив находится внутри исходного байтового массива.

Итак, алгоритм программы memweight следующий. Сначала последовательность байтов до границы слова обрабатывается вызовами hweight8. Далее запускается bitmap_weight. Оставшиеся в конце байты обрабатываются в цикле через hweight8.

Оператор **assert**, поставленный вместо оператора BUG_ON, должен превратиться в условие корректности.

Функция bitmap_weight имеет два параметра: src – указатель на массив слов, nbits – число битов шкалы, представляющей множество. Функция bitmap_weight является интерфейсной в библиотеке ОС Linux и может быть вызвана пользователем независимо. Значение nbits может быть не кратно 8. Код функции приведен ниже.

```
BITMAP_LAST_WORD_MASK(nbits) = (~0UL >> (-(nbits) & (BITS_PER_LONG - 1)));

int bitmap_weight(const unsigned long *src, unsigned int nbits)
{
        if (nbits <= BITS_PER_LONG && nbits > 0)
                return hweight_long(*src & BITMAP_LAST_WORD_MASK(nbits));
        return __bitmap_weight(src, nbits);
}
```

В функции bitmap_weight вызывается hweight_long в случае, когда массив состоит из одного слова. В общем случае реализуется вызов __bitmap_weight с теми же параметрами.

Функция __bitmap_weight аналогична bitmap_weight. Каждое слово обрабатывается вызовом функции hweight_long. Код и описание функции доступны по адресу: https://elixir.bootlin.com/linux/v5.7.8/source/lib/bitmap.c#L333

```
int __bitmap_weight(const unsigned long *bitmap, unsigned int bits)
{
        unsigned int k, lim = bits/BITS_PER_LONG;
        int w = 0;
        for (k = 0; k < lim; k++)
                w += hweight_long(bitmap[k]);
        if (bits % BITS_PER_LONG != 0)
                w += hweight_long(bitmap[k] & BITMAP_LAST_WORD_MASK(bits));
        return w;
}
```

Программа hweight_long определяет число единиц в слове w, являющимся параметром. В зависимости от архитектуры вызывается hweight32(w) или hweight64(w). В этих функциях применяются нетривиальные алгоритмы битовой магии [4]. Ниже пример одной из таких функций.

```
unsigned long hweight64No(__u64 w)
{
        __u64 res = w - ((w >> 1) & 0x5555555555555555ul);
        res = (res & 0x3333333333333333ul) + ((res >> 2) & 0x3333333333333333ul);
        res = (res + (res >> 4)) & 0x0F0F0F0F0F0F0F0Ful;
        res = res + (res >> 8);
        res = res + (res >> 16);
        return (res + (res >> 32)) & 0x00000000000000FFul;
}
```

3.2. Устранение указателей

После этапа нормализации программы применяется набор трансформаций, устраняющих указатели в программе. Доступ по указателю заменяется явным обращением к элементу массива. Арифметические действия с указателем заменяется пересчетом индекса массива.

Указателю ptr функции memweight соответствует массив байтов. Другой указатель bitmap смотрит внутрь этого массива. Обычно при наличии нескольких переменных-указателей, смотрящих внутрь одного массива, одна из переменных становится массивом, а другие – индексами в этом массиве [6, 7]. Однако в данном случае используется более общая схема трансформации, когда все указатели заменяются индексами массива памяти программы. Причина этого – использование операции «%» остатка от деления в применении к указателю bitmap.

Вначале процесса трансформации определяются новые типы и глобальные переменные. Введем тип Mem для глобального массива байтов mem, определяющего всю память.

```
type Mem = array(char, size_t);
Mem mem;
```

Указатель ptr становится индексом в массиве mem типа size_t, соответствующего размеру адресуемой памяти для текущей архитектуры ЭВМ. Вместо указателя bitmap введем переменную p (типа size_t), определяющую индекс элемента в массиве mem, который соответствует указателю bitmap.

Таким образом, программа memweight вычисляет число битов, содержащихся в вырезке mem[ptr..ptr+bytes-1]. Необходимым дополнительным требованием должно быть условие того, что данный массив не выходит за границы максимально адресуемой памяти:

```
ptr+bytes <= SIZE_MAX
```

Указатель (**unsigned long** *)bitmap есть указатель на массив слов. Этот указатель получен операцией приведения типа (**type cast)**. Данный массив слов определим глобальной переменной memL:

```
type MemL = array(unsigned long, size_t);
MemL memL;
```

Массив memL является частью массива mem. Элемент memL[0] начинается с ближайшей границы слова в массиве mem.

Ниже трансформированная программа memweight. Удален первый параметр вызова bitmap_weight, поскольку параметр – массив memL является глобальной переменной.

```
size_t memweight(size_t ptr, bytes)
{
        size_t ret = 0;
        size_t longs;
        size_t p = ptr;
        for (; bytes > 0 && p % sizeof(long) != 0;
                        bytes--, p++)
                ret += hweight8(p);
        longs = bytes / sizeof(long);
        if (longs != 0) {
                assert(longs < INT_MAX / BITS_PER_LONG);
                ret += bitmap_weight(longs * BITS_PER_LONG);
                bytes -= longs * sizeof(long);
                p += longs * sizeof(long);
        }
        for (; bytes > 0; bytes--, p++)
                ret += hweight8(p);
        return ret;
}
```

Далее представлены трансформации функций bitmap_weight и __bitmap_weight. Первый параметр опускается. Вхождение *src в bitmap_weight заменяется на memL.

```
int bitmap_weight(unsigned int nbits)
{
        if (nbits <= BITS_PER_LONG && nbits > 0)
                return hweight_long(memL[0] & BITMAP_LAST_WORD_MASK(nbits));
        return __bitmap_weight(nbits);
}
```

Вхождения указателя bitmap в функции __bitmap_weight заменяются на memL.

```
int __bitmap_weight(unsigned int bits)
{
        unsigned int k, lim = bits/BITS_PER_LONG;
        int w = 0;
        for (k = 0; k < lim; k++)
                w += hweight_long(memL[k]);
        if (bits % BITS_PER_LONG != 0)
                w += hweight_long(memL[k] & BITMAP_LAST_WORD_MASK(bits));
        return w;
}
```

3.3. Трансформация в предикатную программу

Программа, полученная устранением указателей, остается в рамках языка Си. Ее, в принципе, можно верифицировать традиционным методом Хоара.

Преобразование данной программы на язык предикатного программирования P является несложным и в принципе может быть проведено автоматически.

Преобразуем циклы в рекурсивные программы. Имеется три цикла:

```
for (; bytes > 0 && p % sizeof(long) != 0; bytes--, p++)
        ret += hweight8(mem[p]);

for (; bytes > 0; bytes--, p++)
        ret += hweight8(mem[p]);

for (k = 0; k < lim; k++)
        w += hweight_long(memL[k]);
```

Построим для них рекурсивные программы.

```
bits1(bytes, p, ret0 : bytes', p', ret') {
        if (bytes > 0 && p % sizeof(long) != 0)
                bits1(bytes-1, p+1, ret0 + hweight8(mem[p]) : bytes', p', ret')
        else  { bytes' = bytes ||  p' = p ||  ret' = ret0 }
};

bits2(bytes, p, ret0 : ret') {
        if (bytes > 0)
                bits2(bytes-1, p+1, ret0 + hweight8(mem[p]) : ret')
        else ret' = ret0
};
```

```
weightL(k, w0, lim : k', w') {
        if (k < lim)
                weightL(k+1, w0 + hweight_long(memL[k])) : k', w')
        else { w' = w0 || k' = k }
}
```

Замена циклов на вызовы рекурсивных программ дает следующие программы:

```
size_t memweight(size_t ptr, bytes)
{
        bits1(bytes, ptr, 0 : bytes1, p, ret);
        size_t longs = bytes1 / sizeof(long);
        if (longs != 0) {
                assert(longs < INT_MAX / BITS_PER_LONG);
                ret1 = ret + bitmap_weight(longs * BITS_PER_LONG) ||
                bytes2 = bytes1 - longs * sizeof(long) ||
                p1 = p + longs * sizeof(long)
        } else { bytes2 = bytes1 || p1 = p || ret1 = ret }
        bits2(bytes2, p1, ret1 : ret2);
        return ret2;
}

int __bitmap_weight(unsigned int bits)
{
        unsigned int lim = bits/BITS_PER_LONG;
        weightL(0, 0 : k, int w);
        if (bits % BITS_PER_LONG != 0)
                w += hweight_long(memL[k] & BITMAP_LAST_WORD_MASK(bits));
        return w;
}
```

4. Спецификация предикатной программы

Предикатная программа memweight содержит набор глобальных типов, констант и переменных и включает набор функций memweight, bitmap_weight, __bitmap_weight, bits1, bits2 и weightL. Спецификации программы определяют предусловия и постусловия функций в дополнении к типам аргументов и результатов функций.

Сначала определим типы языка Си.

```
const int wsize;  // =sizeof(long); = 4 для 32 бит, =8 для 64 бит
type int_t = INT_MIN .. INT_MAX;
type uint_t = 0 ..  UINT_MAX;
type long_t = LONG_MIN ..  LONG_MAX;
type ulong_t = 0 .. ULONG_MAX;
type size_t = 0 ..  SIZE_MAX;
```

Значения констант для граничных значений типов зависят от архитектуры ЭВМ.

Определим тип Mem для глобального массива байтов mem, определяющего всю память.

```
type Mem = array(char, size_t);
Mem mem;
size_t   ptr, bytes;
```

Переменные ptr и bytes, параметры функции memweight, определены здесь как глобальные переменные. Перевычисляемые значения переменной bytes внутри функции memweight определены другими переменными.

Таким образом, исходная область памяти, в которой нужно определить число единичных битов, определяется вырезкой mem[ptr..ptr+bytes-1]. Эта область памяти разделяется на три части: короткий байтовый массив до первой границы слова, словный массив и короткий байтовый массив за последним словом. Словный массив находится внутри исходного байтового массива. Эквивалентом данного словного массива является массив memL.

Определим формализацию разделения на три части исходной области памяти, что позволит в дальнейшем существенно упростить доказательство формул корректности. Введем глобальные переменные:

```
size_t   ptrB, ptrU, ptrW, longW, bytesU, bytesW;
```

Здесь ptrB – ближайший к ptr индекс границы слова; ptrU = min(ptrB, ptr+bytes); ptrW – индекс байта в массиве mem за последним словом внутри вырезки mem[ptr..ptr+bytes-1]; longW – число слов внутри вырезки от позиции ptrU. Переменные bytesU и bytesW определяют оставшееся число байтов для позиций ptrU и ptrW в массиве mem. Обычно ptrU = ptrB. Однако если граница слова не достигается внутри вырезки mem[ptr..ptr+bytes-1], то ptrU < ptrB, и тогда ptrU соответствует концу исходной области памяти. Описанная модель представлена на Рис.1 и Рис.2.

Рис.1. Модель разбиения памяти. Общий случай: ptrU = ptrB

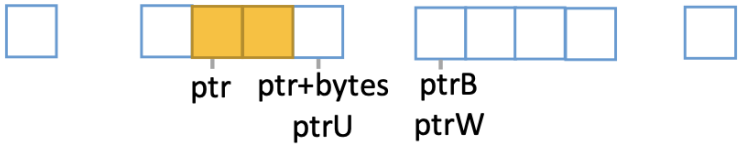


Рис.2. Модель разбиения памяти. Вырожденный случай: ptrU < ptrB

Формально, модель разбиения памяти представляется следующим набором аксиом.

```
formula upper(size_t  p) = p >= ptr & p % wsize = 0 & p – ptr < wsize;
axiom upper(ptrB);
axiom if ptr+bytes< ptrB then ptrU = ptr+bytes else ptrU = ptrB;
axiom bytesU = bytes – (ptrU – ptr);
axiom longW = bytesU / wsize ;
axiom ptrW = ptrU + longW * wsize;
axiom bytesW = bytes – (ptrW – ptr);
```

Представим функции bytew и memw, используемые далее в спецификациях.

```
formula bytew(char b : size_t);
formula memw(size_t  p, n : size_t);
```

Функция bytew определяет число единичных битов в байте b. Функция memw вычисляет число единичных битов вырезки mem[p..p+n–1].

Часть формул, используемых в спецификациях, вводятся без определений. Они будут определены на языке спецификаций why3 с использованием библиотечных функций. Формулы на языке P [3] не всегда адекватно выразимы в языке why3.

Функция memweight вычисляет число единичных битов, содержащихся в вырезке mem[ptr..ptr+bytes-1]. Ниже приводится код функции и постусловие. Отметим, что принадлежность результата ret типу size_t неявно считается частью постусловия.

```
memweight( : size_t ret)
post ret = memw(ptr, bytes)
{
        bits1(bytes, ptr, 0 : size_t bytes1, p1, ret1);
        size_t longs = bytes1 / wsize;
        if (longs != 0) {
                size_t ret2 = ret1 + bitmap_weight(longs * BITS_PER_LONG) ||
                size_t bytes2 = bytes1 - longs * wsize ||
                size_t p2 = p1 + longs * wsize
        } else { size_t  bytes2 = bytes1 || size_t p2 = p1 || size_t ret2 = ret1 } ;
        bits2(bytes2, p2, ret2 : ret)
}
```

Следует отметить, что ограничение ptr+bytes <= SIZE_MAX не может быть проверено при верификации. Оператор **assert** переписывается в виде:

```
theorem bytesU / wsize < INT_MAX / BITS_PER_LONG;
```

Однако данное условие недоказуемо.

В соответствии с определенной выше моделью массив memL является массивом слов длиной longW.

```
type MemL = array(ulong, longW);
MemL memL;
```

Массив memL тождественен вырезке mem[ptrU… ptrU + longW*wsize–1]. Элемент массива memL[k] эквивалентен вырезке mem[ptrU+k*wsize .. ptrU+k*wsize+7].

Ниже приведены коды программ bitmap_weight и __bitmap_weight вместе со спецификациями. В постусловии используется формула bitmw, определяющая число единичных битов в начальной части массива memL длиной nbits бит.

```
formula bitmw (nat nbits: nat);
formula BITMAP_LAST_WORD_MASK(nat nbits: nat) =
(~0UL >> (-(nbits) & (BITS_PER_LONG - 1)));

bitmap_weight(uint_t nbits : int_t ret)
pre nbits / BITS_PER_LONG <= longW
post ret = bitmw(nbits)
{
        if (nbits <= BITS_PER_LONG && nbits > 0)
                ret = hweight_long(memL[0] & BITMAP_LAST_WORD_MASK(nbits));
        else   ret = __bitmap_weight (nbits);
}

__bitmap_weight(uint_t bits: int_t ret)
pre nbits / BITS_PER_LONG <= longW
post ret = bitmw(bits)
{
        uint_t lim = bits/BITS_PER_LONG;
        weightL(0, 0, lim : uint_t k, int_t w);
        if (bits % BITS_PER_LONG != 0)
                w2 = w + hweight_long(memL[k] & BITMAP_LAST_WORD_MASK(bits));
        else w2 = w;
        ret = w2;
}
```

Далее представлены коды и спецификации функций bits1, bits2 и weightL, введенных взамен циклов исходной программы на языке Си. В процессе доказательства формул корректности функции bits1 предусловие было дополнено двумя конъюнктами.

```
bits1(size_t bytes0, p, ret0 : size_t bytes1, p1, ret)
pre ptr<=  p <= ptrU & p+bytes0 = ptr+bytes & ret0 = memw(ptr, p-ptr)
post ret1 = memw(ptr, p1-ptr) & p+bytes0 = p1+bytes1 & bytes1 = bytesU & p1 = ptrU
{
        if (bytes0 > 0 && p % wsize != 0)
                bits1(bytes0-1, p+1, ret0 + hweight8(mem[p]) : bytes1, p1, ret)
        else  { bytes1 = bytes0 ||  p1 = p ||  ret = ret0 }
};

bits2(size_t bytes, size_t p, size_t ret0 : size_t ret)
pre ret0 = memw(ptr, p-ptr)
post ret = ret0 + memw(p, bytes)
{
        if (bytes > 0)
                bits2(bytes-1, p+1, ret0 + hweight8(mem[p]) : ret)
        else ret = ret0
};
```

В спецификации функции weightL использована формула diapw, определяющая число единичных битов в вырезке memL[a .. b-1]. Предусловие функции weightL было уточнено в процессе доказательства формул корректности.

```
formula diapw (nat a, b: nat);

weightL(uint_t k, int_t w0, uint_t lim : uint_t k1, int_t w)
pre k <= lim < longW & w0 = diapw(0, k)
post w = diapw(0, lim) & k1=lim
{
        if (k < lim)
                weightL(k+1, w0 + hweight_long(memL[k])) : k1, w)
        else { w = w0 || k1 = k }
}

formula valL(nat k) = mem[k*wsize .. k*wsize+7];
axiom forall k=0..(longW-1). memL[k] = valL(k);
```

Количество единиц в битовом представлении слова w:

```
formula wordw (ulong w: nat);
```

Формула bitw определяет число бит в вырезке, начинающейся с mem[p] длиной bits битов. Точнее, это вырезка mem[p]...mem[p+bits/8] и плюс возможно в (bits % 8) битах следующего байта, если bits % 8 != 0.

```
formula bitw(size_t  p, uint_t  bits: int_t) =
        if bits >= 8 then memw(p, bits/8*8) + bitw8(p + bits/8*8, bits%8)
        else bitw8(p, bits)
```

Формула bitw8 определяет число битов в байте mem[p] среди первых bits битов.

```
formula bitw8(nat p, nat bits: int_t);
```

5. Процесс дедуктивной верификации

Для предикатной программы, построены формулы корректности программ относительно их спецификаций. Спецификации, определяющие типы, глобальные константы и переменные, формулы предусловий и постусловий, а также формулы корректности оформляются в виде теорий на языке P [3]. Разделение на теории позволяет уменьшить сложность, локализовать вспомогательные леммы, выделить зависимости, а главное, ускорить доказательство через SMT-решатели.

Далее теории кодируются на языке спецификаций Why3 [19]. Отметим, что некоторые определения заимствуются из библиотеки системы Why3. Теории на языке функционального программирования Why3ML представлены в Приложении 2 и отражают состояние после завершения доказательства, включая дополнительно введенные леммы.

Доказательство формул корректности проводилось в системе Why3 версии 1.3.1 с SMT-решателями Z3 4.8.6, CVC3 2.4.1, CVC4 1.7.

Небольшая часть формул была доказана автоматически, ещё несколько формул были доказаны с помощью простых трансформаций introduce_premises, inline_goal.

Для доказательства сложного утверждения, в частности, требующего доказательства по индукции, оказалось полезным использование аппарата лемма-функций [18]. Здесь строится вспомогательная предикатная программа, спецификация которой совпадает с исходным утверждением.

Многие леммы были доказаны в интерактивном режиме, когда процесс доказательства реализуется пользователем последовательным набором команд (трансформаций): introduce_premises, inline_goal, unfold, destruct, rewrite, split_vc, case, apply, assert, instantiate, induction и других команд.

В целом, процесс доказательства лемм можно описать рекурсивной процедурой:

1. делается попытка доказать с помощью разных SMT решателей.
2. если попытка удачна, то ветвь доказана.
3. в противном случае формула разбивается на более простые части с помощью команд split, destruct и других. Далее каждая из этих ветвей обрабатывается с шага 1. Если же

формула уже достаточно простая, то анализируются выражения, и добавляются подсказки с помощью трансформаций assert и case, далее получившиеся ветви обрабатываются с шага 1.

4. в остальных случаях доказательство проводится в интерактивном режиме.

Если какое-то предположение доказывается и встречается при доказательстве разных лемм, оно обобщается и вводится как отдельная лемма для переиспользования.

Отметим трудность автоматического доказательства с помощью SMT-решателей утверждений, содержащие одновременно операции с битовыми векторами и целыми числами.

Верификация битовых функций «битовой магии» hweight8, hweight32 и hweight64 не проводилась. Их верификация описывается в работе [14].

6. Обзор работ

Простые алгоритмы реализации операций с битовыми векторами неэффективны. Разработке эффективных алгоритмов посвящено множество работ. Сверхэффективные методы реализации, использующие нетривиальные свойства разнообразных числовых представлений, получили название битовой магии [4]. Однако работ по дедуктивной верификации алгоритмов битовой магии крайне мало. Алгоритмы битовой магии на языке Why3ML и их верификация в системе Why3 [19] описываются в работе [14]. Разработана теория битовых векторов на языке спецификаций why3, которая позже была включена в стандартную библиотеку системы Why3.

В работе [14] верифицируется несколько алгоритмов, в том числе на языке SPARK. Алгоритмы вычисления числа битов (веса Хэмминга) для 8-, 32- и 64-битных векторов собраны в библиотеке [13]. Они аналогичны соответствующим библиотечным функциям ядра ОС Linux, используемым в настоящей работе. Детали битовой магии подробно описываются в работах [4, 14]. Кроме того, проведена нетривиальная подгонка к используемым SMT-решателям с рекомендациями, как учитывать эти особенности при спецификации и верификации в системе Why3.

В настоящей работе рассматривалась задача вычисления числа битов в последовательности байтов. При этом верификация алгоритмов вычисления числа батов для 8-, 32- и 64-битных векторов не проводилась.

Следует отметить, что в процессе дедуктивной верификации библиотечных программ ядра ОС Linux наряду с системой Why3[19] ранее применялась также система Coq [15]. Теперь стало возможным проводить доказательство полностью в системе Why3, поскольку,

начиная с версии 1.1.3, в ней появилась полноценная система интерактивного доказательства.

7. Заключение

Проведена обратная трансформация библиотечной программы memweight, вычисляющей число элементов множества, представленного в виде битовой шкалы. Построена модель программы (Разд. 4, рис. 1 и 2), используемая при разработке спецификаций предикатной программы. По программе и спецификациям построены формулы корректности, оформленных в виде набора теорий, перенесенных на язык спецификаций why3. Доказательство проводилось в системе Why3[19]. В процессе верификации уточнялись спецификации. Остались недоказанными две формулы корректности, в которых встречаются операции преобразования битовых векторов в целые числа.

В процессе доказательства формул корректности установлена недоказуемость условия, указанного в операторе **assert**. Это означает, что оператор **assert** мог бы сработать на сверхдлинной последовательности байтов. Однако в процессе длительной эксплуатации ОС Linux подобная ситуация ни разу не случилась. Нами была предложена упрощающая модификация программы memweight без оператора **assert**. Модификация передана разработчикам ядра ОС Linux.

Итоги верификации. Отмечена ошибка переполнения выражения ptr+bytes за границы типа size_t. Недоказуемо условие longs < INT_MAX / BITS_PER_LONG оператора **assert**. Других ошибок не обнаружено.

Список литературы

1. Доказательство правил корректности операторов предикатной программы. [Электронный ресурс]. URL:http://www.iis.nsk.su/persons/vshel/files/rules.zip . (дата обращения 12.08.2020).
2. Каблуков И.В., Шелехов В.И. Реализация оптимизирующих трансформаций в системе предикатного программирования. *Системная информатика*, № 11. Новосибирск, 2017. С. 21-48. Электрон. журн. 2018. http://persons.iis.nsk.su/files/persons/pages/opttransform4.pdf. (дата обращения 12.08.2020).
3. Карнаухов Н.С., Першин Д.Ю., Шелехов В.И. Язык предикатного программирования P. Версия 0.12. Новосибирск, 2013. 28с. [Электронный ресурс]. URL: http://persons.iis.nsk.su/files/persons/pages/plang12.pdf. (дата обращения 12.08.2020).
4. Обстоятельно о подсчёте единичных битов. 2016. [Электронный ресурс]. URL: https://habr.com/ru/post/276957/. (дата обращения 12.08.2020).

5. Шелехов В.И. Верификация и синтез эффективных программ стандартных функций в технологии предикатного программирования. *Программная инженерия*, 2011, № 2. С. 14-21.
6. Шелехов В.И. Верификация предикатной программы бинарного поиска объекта произвольного типа. Новосибирск, 2019. 26с. [Электронный ресурс]. URL: http://persons.iis.nsk.su/files/persons/pages/fsearch2.pdf . (дата обращения 12.08.2020).
7. Шелехов В.И. Верификация предикатной программы пирамидальной сортировки — Новосибирск, 2019. 36с.
8. Шелехов В.И. Дедуктивная верификация программы конкатенации строк с применением обратной трансформации // Знания-Онтологии-Теории (ЗОНТ-19). — Новосибирск, 2019. — 19с. [Электронный ресурс]. URL: http://persons.iis.nsk.su/files/persons/pages/logcflc1.pdf. (дата обращения 12.08.2020).
9. Шелехов В.И. Разработка и верификация алгоритмов пирамидальной сортировки в технологии предикатного программирования. Новосибирск, 2012. 30с. (Препр. / ИСИ СО РАН. № 164).
10. Шелехов В.И. Списки и строки в предикатном программировании. *Системная информатика*, ИСИ СО РАН, Новосибирск. Электрон. журн. 2014, №3. С. 25-43. [Электронный ресурс]. URL: http://persons.iis.nsk.su/files/persons/pages/Listcharing.pdf. (дата обращения 12.08.2020).
11. Шелехов В.И., Чушкин М.С. Верификация программы быстрой сортировки с двумя опорными элементами. *Научный сервис в сети Интернет*. М.: ИПМ им. М.В.Келдыша, 2018. 26с. [Электронный ресурс]. URL: http://persons.iis.nsk.su/files/persons/pages/dqsort.pdf. (дата обращения 12.08.2020).
12. Amani, S., Hixon, A., Chen, Z., Rizkallah, C., Chubb, P., O'Connor, L., Beeren, J., Nagashima, Y., Lim, J., Sewell, T., Tuong, J., Keller, G., Murray, T., Klein, G., Heiser, G.: Cogent: Verifying high-assurance file system implementations // Int. Conference on Architectural Support for Programming Languages and Operating Systems. 2016. pp. 175–188.
13. Counting bits in a bit vector. [Электронный ресурс]. URL: http://toccata.lri.fr/gallery/bitcount.en.html. (дата обращения 12.08.2020).
14. Dross C., Fumex C., Gerlach J., Marché C.. High-Level Functional Properties of Bit-Level Programs: Formal Specifications and Automated Proofs. Research Report RR-8821, Inria Saclay. 2015, pp.52.
15. Kennedy T.R. Using program transformations to improve program translation. *Massachusetts Institute of Technology*. AI Memo No.962, 1897. 38p.
16. Shelekhov V. I. Verification and Synthesis of Addition Programs under the Rules of Correctness of Statements // Automatic Control and Computer Sciences. 2011. Vol. 45, No. 7, P. 421–427.
17. The Coq Proof Assistant. [Электронный ресурс]. URL: https://coq.inria.fr/. (дата обращения 12.08.2020).
18. Volkov, G., Mandrykin, M., Efremov, D.: Lemma functions for Frama-C: C programs as proofs. In: Proceedings of the 2018 Ivannikov ISPRAS Open Conference (ISPRAS-2018). pp. 31–38 (2018).

19. Why 3. Where Programs Meet Provers. [Электронный ресурс]. URL:http://why3.lri.fr. (дата обращения 12.08.2020).

Приложение 1. Полный исходный текст программы memweight.c на языке Си

```
#define KERNRELEASE "TEST"
#define BITS_PER_LONG 64
#define INT_MAX 0x7fffffff
#define __stringify_1(x...) #x
#define unlikely(x) __builtin_expect(!!(x), 0)
#define unreachable() do { __builtin_unreachable(); } while (0)
#define __stringify(x...) __stringify_1(x)
#define BUG() do { unreachable(); } while (0)
#define BUG_ON(condition) do { if (unlikely(condition)) BUG(); } while (0)
#define small_const_nbits(nbits) \
        (__builtin_constant_p(nbits) && (nbits) <= BITS_PER_LONG && (nbits) > 0)
#define BITMAP_FIRST_WORD_MASK(start) (~0UL << ((start) & (BITS_PER_LONG - 1)))
#define BITMAP_LAST_WORD_MASK(nbits) (~0UL >> (-(nbits) & (BITS_PER_LONG - 1)))
#define __always_inline inline
#define hweight8(w)  __arch_hweight8(w)
#define hweight32(w) __arch_hweight32(w)
#define hweight64(w) __arch_hweight64(w)

typedef unsigned long long __u64;

struct bug_entry {
 signed int bug_addr_disp;
 signed int file_disp;
 unsigned short line;
 unsigned short flags;
};

typedef unsigned long __kernel_ulong_t;
typedef __kernel_ulong_t __kernel_size_t;
typedef __kernel_size_t              size_t;
//-----------------------------------------------------------------------

#define CONFIG_ARCH_HAS_FAST_MULTIPLIER 1

/**
 * hweightN - returns the hamming weight of a N-bit word
 * @x: the word to weigh
 *
 * The Hamming Weight of a number is the total number of bits set in it.
```

```
 */

unsigned int __sw_hweight32(unsigned int w)
{
#ifdef CONFIG_ARCH_HAS_FAST_MULTIPLIER
        w -= (w >> 1) & 0x55555555;
        w =  (w & 0x33333333) + ((w >> 2) & 0x33333333);
        w =  (w + (w >> 4)) & 0x0f0f0f0f;
        return (w * 0x01010101) >> 24;
#else
        unsigned int res = w - ((w >> 1) & 0x55555555);
        res = (res & 0x33333333) + ((res >> 2) & 0x33333333);
        res = (res + (res >> 4)) & 0x0F0F0F0F;
        res = res + (res >> 8);
        return (res + (res >> 16)) & 0x000000FF;
#endif
}

unsigned long __sw_hweight64(__u64 w)
{
#ifdef CONFIG_ARCH_HAS_FAST_MULTIPLIER
        w -= (w >> 1) & 0x5555555555555555ul;
        w =  (w & 0x3333333333333333ul) + ((w >> 2) & 0x3333333333333333ul);
        w =  (w + (w >> 4)) & 0x0f0f0f0f0f0f0f0ful;
        return (w * 0x0101010101010101ul) >> 56;
#else
        __u64 res = w - ((w >> 1) & 0x5555555555555555ul);
        res = (res & 0x3333333333333333ul) + ((res >> 2) & 0x3333333333333333ul);
        res = (res + (res >> 4)) & 0x0F0F0F0F0F0F0F0Ful;
        res = res + (res >> 8);
        res = res + (res >> 16);
        return (res + (res >> 32)) & 0x00000000000000FFul;
#endif
}

static __always_inline unsigned long __arch_hweight64(__u64 w)
{
        unsigned long res;

        /*asm (ALTERNATIVE("call __sw_hweight64", POPCNT64, X86_FEATURE_POPCNT)
                        : "="REG_OUT (res)
                        : REG_IN (w));*/
  res = __sw_hweight64(w);

        return res;
}

static __always_inline unsigned int __arch_hweight32(unsigned int w)
```

```
{
        unsigned int res;

        /*asm (ALTERNATIVE("call __sw_hweight32", POPCNT32, X86_FEATURE_POPCNT)
                        : "="REG_OUT (res)
                        : REG_IN (w));*/

   res = __sw_hweight32(w);

        return res;
}

static inline unsigned int __arch_hweight16(unsigned int w)
{
        return __arch_hweight32(w & 0xffff);
}

static inline unsigned int __arch_hweight8(unsigned int w)
{
        return __arch_hweight32(w & 0xff);
}

static __always_inline unsigned long hweight_long(unsigned long w)
{
        return sizeof(w) == 4 ? hweight32(w) : hweight64(w);
}

int __bitmap_weight(const unsigned long *bitmap, unsigned int bits)
{
        unsigned int k, lim = bits/BITS_PER_LONG;
        int w = 0;

        for (k = 0; k < lim; k++)
                w += hweight_long(bitmap[k]);

        if (bits % BITS_PER_LONG)
                w += hweight_long(bitmap[k] & BITMAP_LAST_WORD_MASK(bits));

        return w;
}

static __always_inline int bitmap_weight(const unsigned long *src, unsigned int nbits)
{
        if (small_const_nbits(nbits))
                return hweight_long(*src & BITMAP_LAST_WORD_MASK(nbits));
        return __bitmap_weight(src, nbits);
}
```

```
size_t memweight(const void *ptr, size_t bytes)
{
        size_t ret = 0;
        size_t longs;
        const unsigned char *bitmap = ptr;

        for (; bytes > 0 && ((unsigned long)bitmap) % sizeof(long);
                        bytes--, bitmap++)
                ret += hweight8(*bitmap);

        longs = bytes / sizeof(long);
        if (longs) {
                BUG_ON(longs >= INT_MAX / BITS_PER_LONG);
                ret += bitmap_weight((unsigned long *)bitmap,
                                longs * BITS_PER_LONG);
                bytes -= longs * sizeof(long);
                bitmap += longs * sizeof(long);
        }
        /*
         * The reason that this last loop is distinct from the preceding
         * bitmap_weight() call is to compute 1-bits in the last region smaller
         * than sizeof(long) properly on big-endian systems.
         */
        for (; bytes > 0; bytes--, bitmap++)
                ret += hweight8(*bitmap);

        return ret;
}
```

Приложение 2. Теории на языке Why3ML

```
theory MemweightBase
    use int.Int, int.EuclideanDivision, int.NumOf
    use int.Sum as S
    use bool.Bool
    use array.Array, array.ArraySum, array.Init as ArrayInit
    use map.Const as MapConst
    use bv.BV_Gen as BV_Gen
    use bv.BV8 as BV8
    use bv.BV64 as BV64
    use bv.Pow2int

    constant wsize: int = 8 (* 8 для 64 бит, 4 для 32 бит *)
    constant isize: int = 4
    constant lsize: int = wsize
    constant size_max: int = pow2 (wsize * 8) - 1
    constant int_max: int = pow2 (isize * 8 - 1) - 1
    constant int_min: int = -int_max - 1
    constant uint_max: int = int_max * 2 + 1

    constant long_max: int = pow2 (lsize * 8) - 1
    constant long_min: int = -long_max - 1
    constant ulong_max: int = long_max * 2 + 1

    predicate int_t (v: int) = int_min <= v <= int_max
    predicate uint_t (v: int) = 0 <= v <= int_max
    predicate long_t (v: int) = long_min <= v <= long_max
    predicate ulong_t (v: int) = 0 <= v <= ulong_max
    predicate size_t (v: int) = 0 <= v <= size_max
    predicate char_t (v: int) = 0 <= v <= 255

    type mem_t = array int
    constant mem: mem_t
    constant ptr: int
    constant bytes: int
    axiom PtrType: size_t ptr
    axiom BytesType: size_t bytes
    axiom MemLimit: size_t (ptr + bytes)
    axiom Mem_array_char: forall i: int. size_t i -> char_t mem[i]
    axiom Mem_length: length mem = size_max + 1

    predicate upper (p: int) = p = ptr + (mod (wsize - (mod ptr wsize)) wsize)

    lemma UniUpper: forall x y: int. upper x /\ upper y -> x = y
    lemma PropUpper: forall x: int. upper x  -> (mod x wsize) = 0 /\ x - ptr < wsize
```

```
    constant ptrB: int
    axiom PtrB: upper ptrB

    constant ptrU: int
    axiom PtrU: if (ptr + bytes) < ptrB then ptrU = ptr + bytes else ptrU = ptrB
    constant bytesU: int
    axiom BytesU: bytesU = bytes - (ptrU - ptr)

    constant bits_per_long: int = wsize * 8
    axiom BytesU_limit: (div bytesU wsize) < (div int_max bits_per_long)

    constant longW: int
    axiom LongW: longW = div bytesU wsize
    lemma LongW_limit: longW * wsize <= bytesU
    constant ptrT: int
    axiom PtrW: ptrT = ptrU + longW * wsize
    constant bytesT: int
    axiom BytesW: bytesT = bytes - (ptrT - ptr)

    lemma BytesW_lt_wsize: bytesT < wsize
    lemma PtrW_: ptrU < ptrB -> ptrT = ptrU

    (* количество единиц в битовом представлении байта b *)
    let function bytew (b: int): int
       requires { char_t  b}
    =  numof (BV8.nth (BV8.of_int b)) 0 8
    lemma ByteW: forall x: int. char_t x -> bytew x <= 8

    predicate i_mem (p: int) = ptr <= p < ptr + bytes
    predicate p_memw (p b: int) = b >= 0 /\ ptr <= p /\ p + b <= ptr + bytes (*
разрешить p = ptr + bytes /\ b = 0 *)
    lemma P_memw_imem: forall p b: int. p_memw p b -> b = 0 \/ (i_mem p /\ i_mem
(p+b-1))

    constant memW: array int       (* массив весов байтов из mem *)
    axiom MemW_init: forall i: int.
       i_mem i -> memW[i] = bytew mem[i]
    axiom MemW_size: length memW = length mem (* ptr + bytes *)

    let ghost function memw_i (p: int): int
       requires { i_mem p }
    =   bytew mem[p]

    lemma MemW_memw_i_eq: forall p: int. i_mem p -> memw_i p = memW[p]

    lemma Memw_i_limit1: forall p: int. i_mem p -> bytew mem[p] <= 8
    lemma Memw_i_limit2: forall p: int. i_mem p -> memw_i p <= 8
```

```
   let ghost function memw (p b: int): int
      requires { p_memw p b }
(*   =   S.sum memW.elts p (p+b) (*v1*) *)
   = if b = 0 then 0 else S.sum memW.elts p (p+b) (* v2 super-fast Z3 *)
(*   = sum memW p (p+b) (*v3*) *)

   lemma Sum_Zero: forall f: int -> int, i: int. S.sum f i i = 0
   lemma Memw_ZeroBytes: forall p: int. p_memw p 0 -> memw p 0 = 0

   lemma MemW_def2_eq: forall p b: int. p_memw p b -> memw p b = sum memW p
(p+b)
   lemma MemW_def3_eq: forall p b: int. p_memw p b -> memw p b = S.sum memw_i p
(p+b)
   lemma MemW_eq_sum_memw_i: forall p b: int. p_memw p b -> memw p b = S.sum
memw_i p (p+b)

   let rec lemma _Sum_limit (n: int) (f: int -> int) (limit a: int)
     requires { 0 <= n /\ 0 <= limit /\ 0 <= a /\ (forall i: int. a <= i < a+n -> (f i) <=
limit) }
     ensures { S.sum f a (a+n) <= n * limit }
     variant { n }
   = if n > 0 then _Sum_limit (n-1) f limit a

   let rec lemma _MemW_sum_rec (n: int) (p: int)
     requires { p_memw p n }
     ensures { memw p n <= n*8 }
     variant { n }
   = if n > 0 then _MemW_sum_rec (n-1) p

   lemma MemW_sum: forall p x: int. p_memw p x ->  memw p x <= x*8
   axiom Bytes_size_t_limit: size_t (8 * bytes) (* ??????результат memweight влезет в
size_t *)
   lemma Bytes_sum: bytes = (bytes - bytesU) + (bytesU - bytesT) + bytesT
   lemma Memw_SumParts2: forall p p1 b: int. (p <= p1 < p + b /\ p_memw p b) ->
     let b0 = p1-p in
       memw p b = (memw p b0) + (memw p1 (b-b0))
   lemma Memw_SumParts3: forall p p1 p2 b: int. (p <= p1 <= p2 < p + b /\ p_memw p
b) ->
     let b0 = p1-p in let b1 = p2-p1 in let b2 = b-b0-b1 in
        memw p b = (memw p b0) + (memw p1 b1) + (memw p2 b2)
   lemma Memw_SumParts1: forall p b1 b2: int. (p_memw p b1) /\ (p_memw (p+b1) b2)
/\ (p_memw p (b1+b2)) ->
     (memw p b1) + (memw (p+b1) b2) = memw p (b1+b2)
   lemma SumParts: memw ptr bytes =
      memw ptr (bytes - bytesU) + memw ptrU (bytesU - bytesT) + memw ptrT bytesT

   let ghost function bitw8 (p bits: int): int
      requires { size_t p  /\ 0 <= bits < 8 }
```

```
 =   numof (BV8.nth (BV8.of_int mem[p])) 0 bits

let ghost function bitw (p bits: int): int
   requires { p_memw p (div bits 8) /\ 0 <= bits }
=   if bits >= 8 then
       let bt = div bits 8 in memw p bt + bitw8 (p + bt) (mod bits 8)
   else bitw8 p bits

type memL_t = array int
constant memL: memL_t
axiom MemL_length: length memL = longW
axiom MemL_elem_limit: forall k: int. 0 <= k < longW -> ulong_t memL[k]

let ghost function valL (k: int): int
   requires {  0 <= k < longW }
=   let j = k*wsize in    (*  mem[j .. j+7] *)
   ((((((mem[j]*256 + mem[j+1]) * 256 + mem[j+2]) * 256 + mem[j+3]) * 256 +
            mem[j+4]) * 256+ mem[j+5]) * 256 + mem[j+6]) * 256 + mem[j+7]


axiom MemL: forall k: int. if 0 <= k < longW then memL[k] = valL k else memL[k] = 0
lemma MemL_limit: forall k: int. 0 <= memL[k] <= ulong_max

(* количество единиц в битовом представлении слова  w *)
let function wordw (w: int): int
   requires { ulong_t  w}
=   numof (BV64.nth (BV64.of_int w)) 0 64

axiom BV64_int_conv: forall w: int. ulong_t w -> BV64.to_int (BV64.of_int w) = w
axiom BV64_of_int_Zero: BV64.of_int 0 = BV64.zeros
lemma Wordw_Limit: forall w: int. ulong_t w -> wordw w <= 64

let ghost function memlw (i: int): int
   (*requires { 0 <= i < longW }   ?ломается ли корректность?*)
=   if 0 <= i < longW then wordw memL[i] else 0

let ghost function diapw (a b: int): int
   requires { 0 <= a <= b <= longW /\ a < longW }
=   S.sum memlw a b

lemma Diapw_Zero: forall a: int. 0 <= a < longW -> diapw a a = 0

let function wordwB (w bits: int): int
   requires { ulong_t  w /\ 0 <= bits <= 64}
=   if (bits = 0 || w = 0) then 0
   else numof (BV64.nth (BV64.of_int w)) 0 bits

lemma WordwB_Zero: forall b: int. ulong_t b -> wordwB b 0 = 0
```

```
   lemma WordwB_Limit: forall w bits: int. ulong_t w /\ 0 <= bits <= 64 -> wordwB w
bits <= bits

   let ghost function bitmw (nbits: int): int
      requires { 0 <= nbits <= longW * 64 }
   =   let wholewords = div nbits 64 in let tailbits = mod nbits 64 in
      (S.sum memlw 0 wholewords) + if tailbits = 0 then 0 else (wordwB
(memL[wholewords]) tailbits)

   let rec lemma _Bitmw_whole_limit (n: int)
     requires { 0 <= n <= longW }
     ensures { (S.sum memlw 0 n) <= n*64 }
     variant { n }
   = if n > 0 then _Bitmw_whole_limit (n-1)

   lemma Bitmw_Limit: forall nbits: int. 0 <= nbits <= longW * 64 -> bitmw nbits <=
nbits

   (* simpler implementation *)
   function bitmap_last_word_mask   (nbits: int): BV64.t =
      if nbits = 0 then BV64.ones
      else BV64.lsr BV64.ones (bits_per_long - nbits)

   (* = (~0UL >> (-(nbits) & (bits_per_long - 1)))*)
   function long_mask_nbits(v nbits: int): int =
      (BV64.to_int
         (BV64.bw_and (BV64.of_int v) (bitmap_last_word_mask nbits)))

end (*MemweightBase*)


theory Memweight
   use MemweightBase
   use int.Int, int.EuclideanDivision

   predicate qmemweight(ret: int) = ret = (memw ptr bytes) && ret <= size_max &&
size_t ret

   predicate pbits1 (bytes0 p ret0: int) =
       size_t bytes0 /\ size_t p /\ size_t ret0 /\
       ptr <= p <= ptrU /\ p+bytes0 = ptr+bytes /\ ret0 = memw ptr (p-ptr)
   predicate qbits1 (bytes0 p bytes1 p1 ret1: int) =
       (ret1 = memw ptr (p1-ptr) /\ p+bytes0 = p1+bytes1 /\ bytes1 = bytesU /\ p1 =
ptrU)
         && (size_t bytes1 /\ size_t p1 /\ size_t ret1)

   goal QSB1: pbits1 bytes ptr 0
```

```
   predicate pbits2 (b p ret0: int) =
      size_t b /\ size_t p /\ size_t ret0 /\ p_memw p b /\ ret0 = memw ptr (p-ptr)
   predicate qbits2 (p b ret0 ret: int) =  ret = ret0 + (memw p b) && size_t ret

   predicate pbitmap_weight (nbits: int) =
      uint_t nbits && div nbits bits_per_long <= longW

   predicate qbitmap_weight (nbits ret: int) =
      ret = bitmw nbits && int_t ret  (*bitw ptrT bits*)

   goal QSB0: forall bytes1 p1 ret1 longs: int.
         qbits1 bytes ptr bytes1 p1 ret1 /\ longs = div bytes1 wsize  /\ not (longs = 0)
          ->   pbitmap_weight (longs * bits_per_long)

   lemma RB1a : forall longs: int.    longs = div bytesU wsize ->
      (memw ptr (ptrU - ptr) + bitmw (longs * bits_per_long)) = memw ptr ((ptrU +
(longs * wsize)) - ptr)
   lemma RB2a : forall longs: int.    longs = div bytesU wsize ->
       ((memw ptr (ptrU - ptr) + bitmw (longs * bits_per_long))
             + memw (ptrU + (longs * wsize)) (bytesU - (longs * wsize)))
                = memw ptr bytes

   goal RB1: forall bytes1 p1 ret1 bytes2 p2 ret2 longs tt: int.
      size_t bytes1 /\ size_t p1 /\ size_t ret1 /\ size_t bytes2 /\ size_t p2 /\ size_t ret2 /\
size_t longs /\ size_t tt /\
      qbits1 bytes ptr bytes1 p1 ret1 /\
      longs = div bytes1 wsize /\
      longs <> 0 /\
      qbitmap_weight (longs * bits_per_long) tt /\ (*tt = bitw ptrT (longs * bits_per_long)
*)
      ret2 = ret1 + tt /\
      bytes2 = bytes1 - longs * wsize /\
      p2 = p1 + longs * wsize
      -> pbits2 bytes2 p2 ret2

   goal RB2: forall bytes1 p1 ret1 bytes2 p2 ret2 ret longs tt: int.
      size_t bytes1 /\ size_t p1 /\ size_t ret1 /\ size_t bytes2 /\ size_t p2 /\ size_t ret2 /\
size_t ret /\ size_t longs /\
      qbits1 bytes ptr bytes1 p1 ret1 /\
      longs = div bytes1 wsize /\
      longs <> 0 /\
      qbitmap_weight (longs * bits_per_long) tt /\
      ret2 = ret1 + tt /\
      bytes2 = bytes1 - longs * wsize /\
      p2 = p1 + longs * wsize /\
      qbits2 p2 bytes2 ret2 ret
      -> qmemweight ret
```

```
   goal RB3: forall bytes1 p1 ret1 bytes2 p2 ret2 longs: int.
      size_t bytes1 /\ size_t p1 /\ size_t ret1 /\ size_t bytes2 /\ size_t p2 /\ size_t ret2 /\
size_t longs /\
      qbits1 bytes ptr bytes1 p1 ret1 /\
      longs = div bytes1 wsize /\
      longs = 0 /\
      ret2 = ret1 /\
      bytes2 = bytes1 /\
      p2 = p1
      -> pbits2 bytes2 p2 ret2
   goal RB4: forall bytes1 p1 ret1 bytes2 p2 ret2 ret longs: int.
      size_t bytes1 /\ size_t p1 /\ size_t ret1 /\ size_t bytes2 /\ size_t p2 /\ size_t ret2 /\
size_t ret /\ size_t longs /\
      qbits1 bytes ptr bytes1 p1 ret1 /\
      longs = div bytes1 wsize /\
      longs = 0 /\
      ret2 = ret1 /\
      bytes2 = bytes1 /\
      p2 = p1 /\
      qbits2 p2 bytes2 ret2 ret
      -> qmemweight ret
end (*Memweight*)

theory Bits1
   use MemweightBase, Memweight
   use int.Int, int.EuclideanDivision
   use array.Array

   goal QC3: forall bytes0 p ret0 bytes1 p1 ret: int.
      size_t bytes0 /\ size_t p /\ size_t ret0 /\ size_t bytes1 /\ size_t p1 /\ size_t ret /\
      pbits1 bytes0 p ret0 /\
      ( bytes0 = 0 \/ (mod p wsize) = 0 ) /\
      bytes1 = bytes0 /\ p1 = p /\ ret = ret0
      -> qbits1 bytes0 p bytes1 p1 ret

   predicate qhweight8 (v w: int) = w = bytew v && uint_t w

   goal RB7: forall bytes0 p ret0 tt: int.
      size_t bytes0 /\ size_t p /\ size_t ret0 /\ uint_t tt /\
      pbits1 bytes0 p ret0 /\
      bytes0 > 0 /\
      mod p wsize <> 0 /\
      qhweight8 mem[p] tt
      -> pbits1 (bytes0-1) (p+1) (ret0+tt)

   goal RB8: forall bytes0 p ret0 bytes1 p1 ret tt: int.
      size_t bytes0 /\ size_t p /\ uint_t ret0 /\ size_t bytes1 /\ size_t p1 /\
      size_t ret /\ uint_t tt /\
```

```
        pbits1 (bytes0-1) p ret0 /\
        bytes0 > 0 /\
        mod p wsize <> 0 /\
        qhweight8 mem[p] tt /\
        qbits1 (bytes0-1) (p+1) bytes1 p1 ret
        -> qbits1 bytes0 p bytes1 p1 ret
end (*bits1*)


theory Bits2
    use MemweightBase, Memweight, Bits1
    use int.Int
    use array.Array

    goal QC4: forall b p ret0 ret: int.
        size_t b /\ size_t p /\ size_t ret0 /\ size_t ret /\
        pbits2 b p ret0 /\
        b = 0 /\ ret = ret0
        -> qbits2 p b ret0 ret
    goal RB9: forall b p ret0 tt: int.
        size_t b /\ size_t p /\ size_t ret0 /\  size_t tt /\
        pbits2 b p ret0 /\
        b > 0 /\
        qhweight8 mem[p] tt
        -> pbits2 (b-1) (p+1) (ret0+tt)
    goal RB10: forall b p ret0 ret tt: int.
        size_t b /\ size_t p /\ size_t ret0 /\ size_t ret /\ uint_t tt /\
        pbits2 b p ret0 /\
        b > 0 /\
        qhweight8 mem[p] tt /\
        qbits2 (p+1) (b-1) (ret0 + tt) ret
        -> qbits2 p b ret0 ret
end (*bits2*)


theory WeightL
    use MemweightBase
    use int.Int, int.NumOf, int.EuclideanDivision
    use array.Array

    predicate pweightL (w0 k lim: int) = int_t w0 /\ uint_t k /\ k <= lim /\ lim < longW /\
w0 = diapw 0 k (*memw ptrU k*wsize*)
    predicate qweightL (w0 lim k1 w: int) = int_t w0 /\ uint_t lim /\ uint_t k1 /\ int_t w
            -> w = diapw 0 lim /\ k1 = lim (*w = w0 + (memw ptrT (lim * wsize)) /\
k1=lim*)
    predicate qhweight_long (w ret: int) = ulong_t w ->
            ret = numof (BV64.nth (BV64.of_int w)) 0 (wsize * 8) && ulong_t ret
```

```
   (* additional restrictions *)
   lemma Bitmw_Eq_Diapw_longs: forall n: int. 0 <= n < longW ->
      bitmw (n * bits_per_long) = diapw 0 n
   lemma Bitmw_Eq_Diapw_bits: forall bits longs nbits: int.
      0 <= bits < longW * bits_per_long /\ nbits = (mod bits bits_per_long) /\ longs =
(div bits bits_per_long)
     ->
      longs <= longW /\
      bitmw bits = (diapw 0 longs) + if nbits = 0 then 0 else (wordwB memL[longs] nbits)

   let rec lemma _DiapW_Bitmw_eq (b: int)
    requires { 0 <= b < longW }
    ensures { diapw 0 b = bitmw (b * bits_per_long) }
    variant { b }
   = if (0 < b) then _DiapW_Bitmw_eq (b-1)

   axiom Diapw_Limit_Int: forall k: int. 0 <= k < longW -> int_t (diapw 0 k)
   lemma Diapw_Step: forall n: int. 0 <= n < longW-1 -> diapw 0 n + wordwB memL[n]
(wsize * 8) = diapw 0 (n+1)
   lemma Wordw_eq_wordwB: forall w: int. ulong_t w -> wordw w = wordwB w (wsize *
8)
   lemma Qhweight_long_eq_wordw: forall w: int. ulong_t w -> qhweight_long w (wordw
w)
   (* end additional restrictions *)


   goal QC5: forall k w0 lim k1 w: int.
      uint_t k /\ int_t w0 /\ uint_t lim /\ uint_t k1 /\ int_t w /\
      pweightL w0 k lim /\
      k >= lim /\
      w = w0 /\ k1 = k
      -> qweightL w0 lim k1 w
   goal RB11: forall k w0 lim tt: int.
      uint_t k /\ int_t w0 /\ w0 = diapw 0 k /\ uint_t lim /\ int_t tt /\
      pweightL w0 k lim /\
      k < lim /\
      qhweight_long memL[k] tt
      -> pweightL (w0+tt) (k+1) lim
   goal RB12: forall k w0 lim k1 w tt: int.
      uint_t k /\ int_t w0 /\ uint_t lim /\ uint_t k1 /\ int_t w /\ int_t tt /\
      pweightL w0 k lim /\
      k < lim /\
      qhweight_long memL[k] tt /\
      qweightL (w0+tt) lim (k+1) w
      -> qweightL w0 lim k w
end  (*weightL*)
```

```
theory A__bitmap_weight
  use MemweightBase, WeightL
  use int.Int, int.EuclideanDivision, int.NumOf
  use array.Array

  predicate p__bitmap_weight (bits: int) =
    uint_t bits && div bits bits_per_long <= longW

  predicate q__bitmap_weight (bits ret: int) =
    ret = bitmw bits && int_t ret (*bitw ptrT bits*)

  goal QSB2: forall bits lim: int.
    uint_t bits /\ uint_t lim /\
    lim = div bits bits_per_long /\ lim < longW /\ longW > 0
    -> pweightL 0 0 lim
  goal QC2: forall bits lim w k ret: int.
    uint_t bits /\ uint_t lim /\ int_t w /\ uint_t k /\ int_t ret /\
    lim = div bits bits_per_long /\ lim < longW /\
    qweightL 0 lim k w /\
    mod bits bits_per_long = 0 /\
    ret = w
    -> q__bitmap_weight bits ret

  axiom WordwB_eq_mask: forall w b: int. ulong_t w /\ 0<=b<=bits_per_long ->
    wordw (long_mask_nbits w b) = wordwB w b
  lemma Qhweight_long_eq_wordwB: forall w tt: int. ulong_t w /\ qhweight_long w tt ->
tt = wordw w
  lemma Qhweight_long_eq_wordwB2: forall w tt: int. ulong_t w -> numof (BV64.nth
(BV64.of_int w)) 0 (wsize * 8) = wordw w
  lemma Long_mask_nbits_Limit: forall w, nbits: int. ulong_t w /\ 0 <= nbits <=
bits_per_long -> long_mask_nbits w nbits <= w

  goal COR1: forall bits lim k w tt ret nbits: int.
    uint_t bits /\ uint_t lim /\ uint_t k /\ int_t w /\ ulong_t tt /\ uint_t ret /\
    lim = div bits bits_per_long /\ lim < longW /\
    qweightL 0 lim k w /\
    nbits = mod bits bits_per_long /\ nbits <> 0 /\
    qhweight_long (long_mask_nbits memL[k] nbits) tt /\          (* memL[k] &
bitmap_last_word_mask(bits) *)
    ret = w + tt
    -> q__bitmap_weight bits ret
end (*A__bitmap_weight*)

theory Bitmap_weight
  use MemweightBase, Memweight, WeightL, A__bitmap_weight
  use int.Int, int.EuclideanDivision
  use int.NumOf
  use array.Array
```

```
   predicate lessLong (nbits: int) = 0 < nbits <= bits_per_long

   goal RB5: forall nbits ret: int.
      uint_t nbits /\ int_t ret /\
      div nbits bits_per_long <= longW /\
      lessLong nbits /\
      qhweight_long (long_mask_nbits memL[0] nbits) ret (* memL[0] &
bitmap_last_word_mask(nbits) *)
      -> qbitmap_weight nbits ret
   goal RB6: forall nbits ret: int.
      uint_t nbits /\ int_t ret /\
      not lessLong nbits /\
      q__bitmap_weight nbits ret
      -> qbitmap_weight nbits ret
end (*bitmap_weight*)
```

004.032.26

Выделение именованных сущностей из текстов распорядительных документов с помощью глубоких нейронных сетей

Березин С.А. (Московский физико-технический институт, Новосибирский государственный университет),

Бондаренко И.Ю. (Новосибирский государственный университет)

Выделение именованных сущностей - это задача извлечения из текстовых данных информации, принадлежащей к заранее определенным категориям, таким как названия организаций, топонимы, имена людей и т.п. В рамках представленной работы был разработан подход, развивающий идеи предшественников по дообучению глубоких нейроных сетей с механизмом внимания архитектуры BERT. Показано, что предварительное обучение языковой модели задачам восстановления маскированного слова и определению семантической связанности двух предложений позволяет заметно улучшить показатели качества решения задачи выделения именованных сущностей. Достигнут один из лучших результатов в задаче выделения именованных сущностей на наборе данных RuREBus, содержащем тексты распорядительных документов министерства экономического развития Российской Федерации. Одной из ключевых особенностей описываемого решения является близость постановки к реальным бизнес-задачам и выделение сущностей не общебытового характера, а специфичных для экономической отрасли.

Ключевые слова: *глубокие нейронные сети, обработка естественного языка, выделение именованных сущностей, BERT*

1. Введение

Выделение именованных сущностей (NER) – это задача выделения в тексте слов или их сочетаний, обозначающих объект или явление определённой категории, например, названия организаций, имена людей и т.д. [1]. Часто выделенные объекты связаны семантическими отношениями (например: «спрос вырос») – обнаружение таких отношения составляет задачу выделения отношений (RE).

Будучи интуитивно понятными для людей, эти задачи долгое время находились за пределами возможностей автоматизированных систем. Многие годы лучшие решения основывались на некоем наборе правил, составленных вручную или автоматически. Значительным прорывом стало использование рекуррентных нейронных сетей [18, 19] и моделей, основанных на векторных представлениях слов, таких как word2vec [2] и GloVe [3]. Тем не менее, качественный скачок в решении этой задачи произошел только с появлением языковых моделей, использующих механизм внимания [4].

Решение поставленной задачи для русского языка существенно усложняется малым числом размеченных наборов данных и, более того, существующие корпуса данных достаточно далеки от типичной бизнес-постановки задачи по следующим причинам:

1) Во-первых, отношения выделены в тексте достаточно плотно (напротив, в бизнес-задачах часто присутствуют лишь 1-2 вхождения отношений на достаточно объемные тексты).

2) Во-вторых, в стандартных корпусах определены отношения бытового и повседневного характера (отношение работы персоны в компании, купли/продажи, владения, родственные отношения, факты рождения и смерти и т. п.), тогда, как в бизнес-задачах обычно требуется выделять отношения, имеющие специфическую природу, связанную с тематикой предметной области.

1.1 Обзор существующих решений

Отправной точкой для исследования задачи выделения именованных сущностей в текстах на русском языке можно считать работу [6], в которой авторы представляют стандартизированный набор данных для обучения алгоритмов выделения именованных сущностей и описывают несколько базовых подходов, послуживших основой для дальнейших работ.

Участники прошедшего в 2016 году соревнования FactRuEval-2016 [14], посвященного задаче выделения именованных сущностей, предложили несколько решений данной проблемы. Так, Сысоев А.А. и Адрианов И.А. предложили подход, основанный на использовании языковой модели word2vec [2] – будучи обученной на задаче предсказания пропущенного слова по окружающим его словам (или предсказания окружающих слов по данному) такая модель способна сформировать семантически значимые векторные представления слов.

В начале 2019 года была опубликована работа [7], в которой авторы описывают подход, основанный на архитектуре CharCNN-BLSTM-CRF, и достигают примечательных результатов.

В том же году, в ходе соревнования BSNLP-2019 [13] выдающиеся результаты были показаны Tsygankova et al., использовавшими подход на основе BiLSTM-CRF [15] с применением эмбеддингов, полученных неизменённой языковой моделью BERT multilingual, и Arkhipov et al., которые имплементировали модифицированную версию архитектуры BERT, путем дообучения мультиязычной архитектуры на текстах на русском языке, скомбинированную с CRF в качестве выходного слоя [16].

2. Описание данных

Для обучения использовался корпус Минэкономразвития, который представляет собой различные отчеты региональных органов о проделанной работе и запланированных мероприятиях, а также прогнозы и планы на будущее. Некоторое подмножество корпуса заранее размечено специальными именованными сущностями (8 классов) и семантическими отношениями на них (11 классов).

По своей постановке задача является задачей классификации слов. В постановке задачи присутствуют (не считая тип 0 – прочие слова) следующие типы выделяемых сущностей:

1) MET (metric) – численный индикатор\показатель, объект, на котором определена операция сравнения (пример: «производительность труда»);
2) ECO (economics) – экономическая сущность (из тех, что не подходят под определение MET) или объект инфраструктуры (пример: «биологических ресурсов»);
3) BIN (binary) – одноразовое действие\бинарная характеристика (есть или нет) (пример: «создание», «строительство»);
4) CMP (compare) – сравнительная характеристика (пример: «выше чем»);
5) QUA (qualitive) – качественная характеристика (пример: «стабильное»);
6) ACT (activity) — принимаемые меры, проводимые мероприятия (пример: «профилактика наркомании»);
7) INST (institutions) — различные учреждения, заведения, структуры и организации (пример: «центр досуга и творчества»);
8) SOC (social) – социальный объект (пример: «кадровая система»)

Разметка корпуса выполнена в формате brat standoff [17] и включает в себя номер объекта или отношения, тип, позицию первого и последнего символов объекта или перечисление аргументов отношения. Пример данных показан на рисунке 1.

```
T127    SOC 6730 6742   правопорядка
T128    BIN 6955 6965   реализации
T129    BIN 7009 7019   достижение
T130    INST 7198 7214  Правительства РК
T131    INST 7387 7403  Правительства РК
R1      GOL Arg1:T3 Arg2:T4
R2      GOL Arg1:T3 Arg2:T5
R3      GOL Arg1:T70 Arg2:T6
R4      GOL Arg1:T70 Arg2:T7
R5      TSK Arg1:T10 Arg2:T11
```

Рисунок 1 – Пример размеченных данных

3. Предобучение языковой модели

Предлагаемые модели для NER основаны на архитектуре BERT [8]. В качестве начальной точки обучения был взят RuBERT [9] — дообученный на текстах русской Википедии и новостных статьях вариант модели BERT. Использование этой, а не мультиязычной модели, как показано далее, даёт заметный прирост в качестве работы. Веса RuBERT использовались в дальнейшем дообучении модели на неразмеченном текстовом корпусе отчетов Минэкономразвития, который описан в предыдущем разделе.

Для обучения модели использовался Google Cloud TPU v2 (tensor processing unit). Размер пакета – 128, скорость обучения – $\mathbf{2 * 10^{-5}}$. Максимальная длина последовательности – 512 токенов, доля маскируемых токенов – 0.15, размер словаря – 119547 токенов. Результаты экспериментов приведены в таблице 1.

Таблица 1 – Результаты экспериментов по обучению BERT

Название модели	Число итераций обучения	Значение loss функции	Результат в задаче NER, F1 macro
BERT Multilingual	0	9.3	0.5458±0.0115
RuBERT	0	9.1	0.555±0.003
BERT Multilingual	210000	2.9	0.430
RuBERT	210000	2.8	0.463

RuBERT	2000	6.2	0.5636
RuBERT	**2500**	5.8	**0.5758**
RuBERT	3000	5.01	0.5732
RuBERT	4000	4.15	0.5332

Как видно в таблице 1, модель довольно быстро переобучается под задачи предсказания маскированного слова и предсказания следующего предложения, что плохо сказывается на её применимости к задаче выделения именованных сущностей.

Лишь спустя несколько экспериментов удалось выявить оптимальное количество итераций обучения. Также, проверялась гипотеза о том, что составление совершенно нового словаря может улучшить показатели, однако, на столь малом объеме текстов модель не смогла хорошо обучиться с нуля, поэтому было принято решение продолжить использование предобученной модели.

Таким образом, была получена модель, адаптированная не только для русского языка в целом, но и конкретно для формального стиля официальных документов.

4. Обучение классификатора

Добавленные к RuBERT слои для решения задачи NER – это классификатор, который принимает векторные представления слов и возвращает категории сущностей, соответствующие этим словам. На вход добавленными слоями принимаются сгенерированные предобученным BERT векторные представления слов, которые затем последовательно принимаются блоком BiLSTM, служащим для выявления зависимостей между токенами предложения. Следующим шагом является полносвязный слой и, наконец, мы используем слой CRF для получения меток классов. Архитектура модели

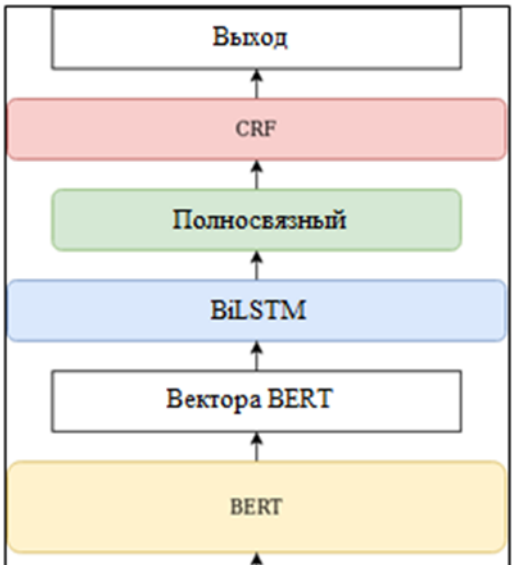
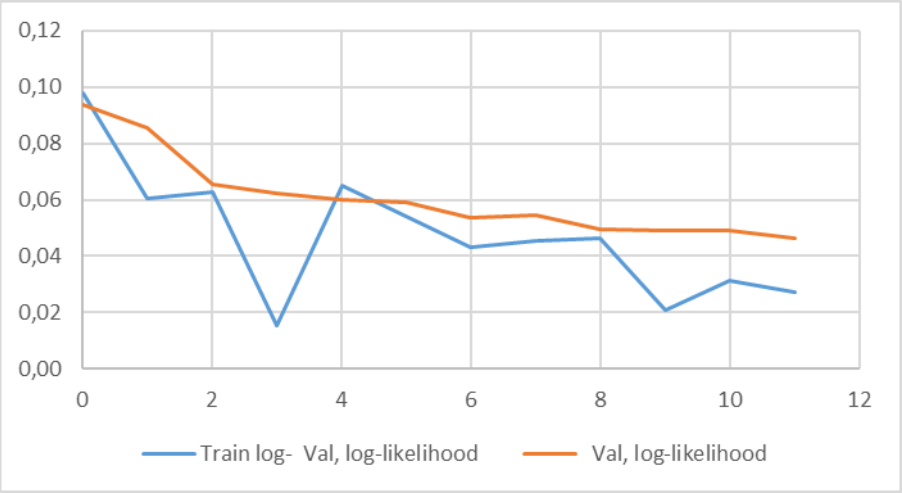


Рисунок 2 – Архитектура нейросети для определения именованных сущностей

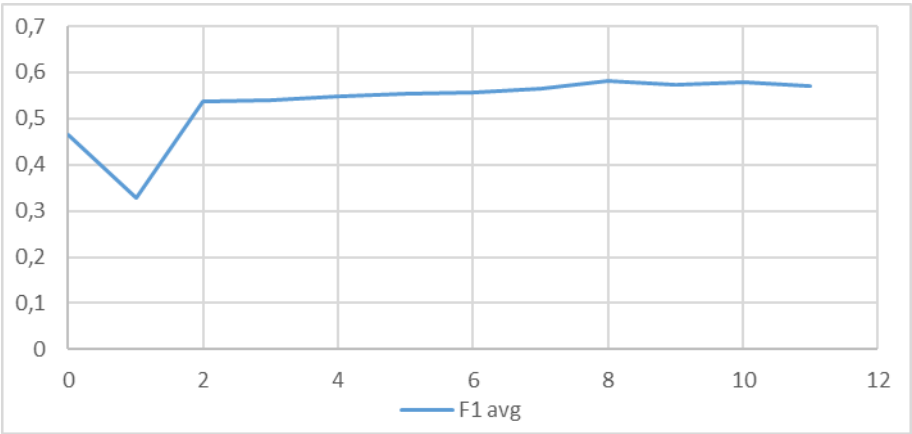


продемонстрирована на рисунке 2.

Рисунок 3 – Значения функции потерь во время обучения

Обучение производилось на GPU Nvidia Tesla P100. Размер мини-выборки составлял 128 экземпляров, скорость обучения равнялась $1 * 10^{-4}$. Для обучения использовалось 32109 примеров, для тестирования 4699 примеров. В качестве меры качества использовалась макро-усреднённая F1-мера. В качестве функции потерь использовалась мультиклассовая кросс-энтропия. Ход эксперимента отображен на рисунках 3-7.

Как видно на рисунке 3, обучение было остановлено раньше, чем в значениях функции потерь на валидационной выборке появились какие-либо признаки переобучения. Причиной этому является неабсолютная корреляция функции потерь с фактической метрикой качества. Если мы обратимся к рисунку 4, то заметим, что F1-мера на валидационной выборке не только вышла на плато, но даже несколько упала, что позволяет говорить о том, что переобучение, де-факто, в некоторой степени уже



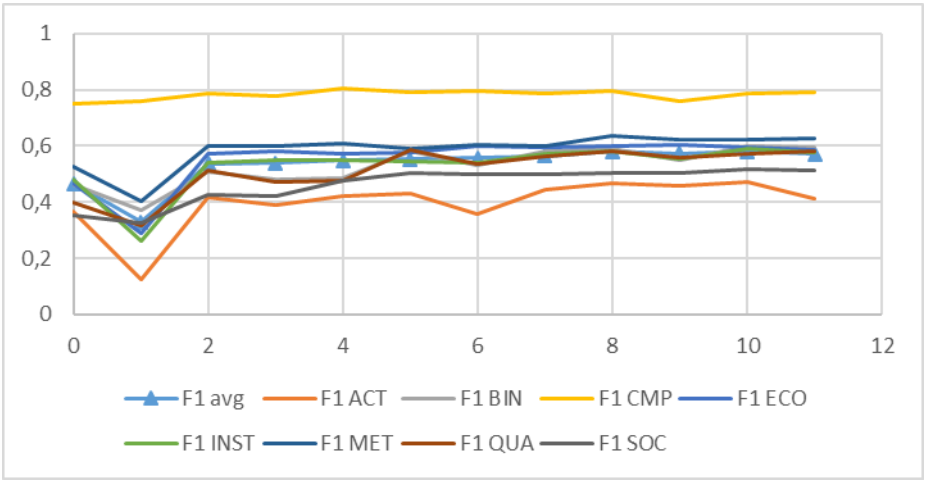


Рисунок 5 – Значения F1-меры во время обучения

произошло.

Рисунок 4 – Значения усреднённой F1-меры во время обучения

Более подробная картина изменения значений F1-меры на валидационных данных продемонстрирована на рисунке 5. Нельзя не обратить внимание на разительно отличающиеся в лучшую сторону показатели качества распознавания сущностей, характеризующих качественные характеристики (QUA). Если ставить задачу выявления только таких сущностей, то F1-мера обученной модели достигает значений вплоть до 0.80.

Также стоит отметить, что, помимо класса QUA, класс социальных объектов (SOC) не оказался подвержен столь значительным флуктуациям, как все остальные классы в начале обучения и стабильно показывал быстрый рост качества вплоть до 5-ой эпохи. Класс принимаемых мероприятий (ACT) стабильно оказывался самым сложным для выделения.

Любопытно также подробнее изучить ход обучения и проанализировать точность и полноту распознавания классов, представленные на рисунках 6 и 7 соответственно.

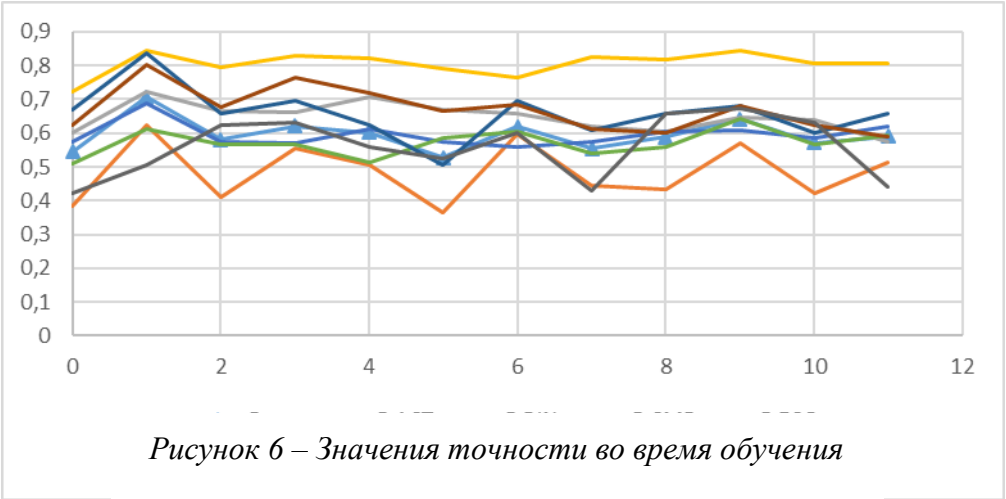


Рисунок 6 – Значения точности во время обучения

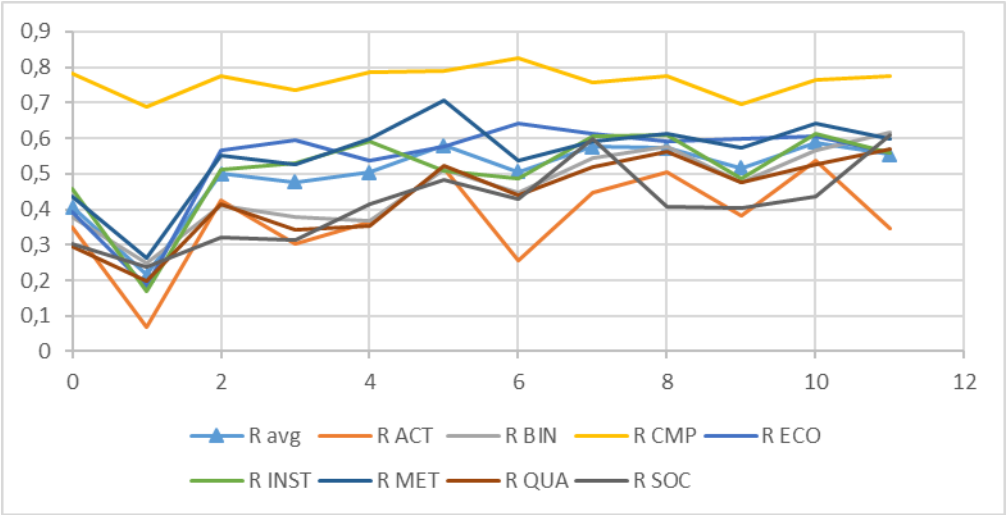


Рисунок 7 – Значения полноты во время обучения

В целом, заметна тенденция модели не отмечать лишних объектов в начале обучения, при этом пропуская много правильных ответов, которая устраняется в дальнейшем.

5. Результаты

В таблице 2 приведено сравнение достигнутых результатов с результатами участников конкурса RuREBus [5], проводимого в рамках конференции Dialog-2020, перед участниками которого ставилась та же задача, что была рассмотрена в ходе данной работы. Важно отметить: несмотря на то, что результат был получен на том же самом тестовом наборе данных, он не был получен во время официальной судейской оценки.

Таблица 2 – Сравнение результатов

Команда	Результат (F1-мера, макро усреднённая)
davletov-aa	0.561
Sdernal	0.464
ksmith	0.463
viby	0.417
dimsolo	0.400
bond005	0.338

Student2020	0.253
Данная работа	**0.576**

6. Заключение

В результате выполненной работы разработано одно из лучших решений задачи выделения именованных сущностей на наборе данных RuREBus.

Продемонстрировано, что предварительное обучение языковой модели задачам восстановления маскированного слова и определению семантической связанности двух предложений позволяет улучшить показатели качества решения задачи выделения именованных сущностей и не требует при этом затрат человеческих усилий на какую-либо разметку и подготовку данных.

В процессе работы над решением были определены следующие пути улучшения:

1) использование более сложных классификаторов – например, байесовской нейронной сети [10];
2) использование более совершенных языковых моделей, таких как ERNIE [11];
3) использование аугментации данных с применением языковой модели для оценки примеров обучения, сгенерированных автоматически из существующих данных [12].

Список литературы

1. Georgios Petasis, Frantz Vichot, Francis Wolinski, Georgios Paliouras Using Machine Learning to Maintain Rule-based Named-Entity Recognition and Classification Systems // ACL '01: Proceedings of the 39th Annual Meeting on Association for Computational Linguistics. 2001. P. 426–433
2. Sysoev A. A., Andrianov I. A. Named Entity Recognition in Russian: the Power of Wiki-Based Approach // Computational Linguistics and Intellectual Technologies: Proceedings of the International Conference “Dialogue 2016”: 2016. URL: http://www.dialog-21.ru/media/3433/sysoevaaandrianovia.pdf (дата обращения: 25.09.2020).
3. Pennington Jeffrey, Socher Richard, Manning Christopher Glove: Global Vectors for Word Representation // Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP): 2014. P. 1532–1543
4. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin Attention is all you need // Proceedings of the 31st International Conference on Neural Information Processing Systems: 2017. P. 6000–6010

5. Ivanin, V., Artemova, E., Batura, T., Ivanov, V., Sarkisyan, V., Tutubalina, E., & Smurov, I. RuREBus-2020 Shared Task: Russian Relation Extraction for Business // Computational Linguistics and Intellectual Technologies: Proceedings of the International Conference “Dialog”. 2020. URL: http://www.dialog-21.ru/media/5098/ivaninvaplusetal-182.pdf (дата обращения: 25.09.2020).
6. Rinat Gareev, Maksim Tkachenko, Valery D Solovyev, Andrey Simanovsky, Vladimir Ivanov Introducing Baselines for Russian Named Entity Recognition // CICLing 2013: Computational Linguistics and Intelligent Text Processing. 2013. P. 329–342
7. Anh Le, Mikhail Burtsev A Deep Neural Network Model for the Task of Named Entity Recognition // International Journal of Machine Learning and Computing vol. 9, no. 1. 2019. URL: http://www.ijmlc.org/vol9/758-ML0025.pdf (дата обращения: 25.09.2020).
8. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // Proceedings 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. P. 4171–4186
9. Kuratov, Y., Arkhipov, M. Adaptation of Deep Bidirectional Multilingual Transformers for Russian Language // Computational Linguistics and Intellectual Technologies: Proceedings of the International Conference “Dialogue 2019”. 2019. URL: http://www.dialog-21.ru/media/4606/kuratovyplusarkhipovm-025.pdf (дата обращения: 25.09.2020).
10. Vikram Mullachery, Aniruddh Khera, Amir Husain. Bayesian Neural Networks // Digital Culture & Society (DCS): Vol. 4. 2018. URL: https://arxiv.org/abs/1801.07710 (дата обращения: 25.09.2020).
11. Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, Qun Liu ERNIE: Enhanced Language Representation with Informative Entities // Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. 2019. P. 1441–1451
12. Jonathan Rotsztejn, Nora Hollenstein, Ce Zhang Effectively Combining ETH-DS3Lab at SemEval-2018 Task 7: Recurrent and Convolutional Neural Networks for Relation Classification and Extraction // Proceedings of The 12th International Workshop on Semantic Evaluation. 2018. URL: https://arxiv.org/pdf/1804.02042.pdf (дата обращения: 25.09.2020).
13. Jakub Piskorski et al., The Second Cross-Lingual Challenge on Recognition, Normalization, Classification, and Linking of Named Entities across Slavic Languages // Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing. 2019. P. 63–74
14. Starostin A. S., Bocharov V. V., Alexeeva S. V., Bodrova A. A., Chuchunkov A. S., Dzhumaev S. S., Efimenko I. V., Granovsky D. V., Khoroshevsky V. F., Krylova I. V., Nikolaeva M. A., Smurov I. M., Toldova S. Y. FactRuEval 2016: Evaluation of Named Entity Recognition and Fact Extraction Systems for Russian // Computational Linguistics and Intellectual Technologies: Proceedings of the International Conference “Dialogue 2016”. 2016. P. 688-705

15. Tatiana Tsygankova, Stephen Mayhew, and Dan Roth BSNLP2019 shared task submission: Multisource neural NER transfer // Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing. Association for Computational Linguistics. 2019. P. 75–82
16. Mikhail Arkhipov, Maria Trofimova, Yuri Kuratov, and Alexey Sorokin Tuning multilingual transformers for language-specific named entity recognition // Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing. 2019. P. 89–93
17. Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou and Jun'ichi Tsujii brat: A Web-based Tool for NLP-Assisted Text Annotation // Proceedings of the Demonstrations Session at EACL 2012. 2012. P. 102–107
18. Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, Chris Dyer Neural Architectures for Named Entity Recognition // Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2016. P. 260–270
19. Xuezhe Ma, Eduard Hovy End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF // Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2016. P. 1064–1074

УДК 81'322.2

Распознавание аргументативных связей в научно-популярных текстах

Саломатина Н.В. (Институт математики им. С.Л. Соболева),
Кононенко И.С. (Институт систем информатики им. А.П. Ершова),
Сидорова Е.А. (Институт систем информатики им. А.П. Ершова),
Пименов И.С. (Новосибирский государственный университет)

В статье представлено исследование эффективности использования признака принадлежности утверждений, участвующих в аргументации, к одному тематическому фрагменту текста. Работа проводилась с целью последующего применения этого признака в автоматическом распознавании аргументационных структур. Объектом исследования служили русскоязычные тексты научно-популярного жанра. Тематическая структура текста строилась на основе выявления сверхфразовых единств (фрагментов текста, объединенных одной темой) путем обнаружения кластеров слов и словосочетаний с помощью сканирующих статистик. Для верификации потенциально возможных связей, извлекаемых из тематической структуры, использовались тексты с ручной разметкой аргументационной структуры. Сопоставление связей, построенных «вручную» и потенциальных, определяемых из тематической структуры, проводилось автоматически. Полученные с макро-усреднением точность и полнота составили 48,6 % и 76,2 % соответственно.

Ключевые слова: *извлечение аргументации,аргументативные связи,тематическая структура текста, профиль кластеризуемости, научно-популярные тексты.*

1. Введение

Область исследований, связанная с автоматическим извлечением аргументов, выделилась в самостоятельную из области тонального анализа текстов, поскольку стала важна не только эмоциональная оценка объектов, явлений и пр. – позитивная, негативная или нейтральная – и мнение о них, но и причины формирования определенных мнений и их обоснование [5]. В настоящее время автоматическое выявление аргументов, аргументационных стратегий в текстах разных жанров полезно, например, для понимания и ведения дебатов, принятия решений (за \ против) в рекомендательных системах, для обнаружения радикальных мнений,

вводящих в заблуждение текстов и пр. В приложениях, реализующих анализ такого типа, структура аргументации должна быть распознана автоматически.

Применение методов машинного обучения для извлечения аргументов и аргументационных структур проводится на базе размеченных специальным образом корпусов текстов. Работа по глубокой разметке корпуса, каковой является разметка аргументации, очень трудоемка, поэтому размеченных текстов часто недостаточно для качественного обучения систем распознавания. Информативность признаков, которые применяются в машинном обучении, в опубликованных работах обычно оценивается в совокупности, поэтому трудно понять, каков вклад того или иного признака в полученном результате. При разработке систем распознавания априорное знание о вкладе отдельных признаков позволит ускорить их отбор для дальнейшего использования.

Автоматическое распознавание структуры аргументов обычно проводится по следующей схеме:

а) выделение утверждений (клауз) в исследуемом тексте;

б) разделение утверждений на аргументативные и неаргументативные;

в) выявление роли утверждений в структуре аргументации – главного тезиса, посылок и заключений;

г) установление связности утверждений с использованием знаний о схемах рассуждений и \ или тематической структуры текста.

Каждый из этапов схемы требует проведения отдельного исследования и часто производится в предположении, что другие этапы некоторым образом реализованы.

Цель работы: оценить вклад тематического признака в решение задачи автоматического построения аргументативных связей между утверждениями на основе размеченного корпуса научно-популярных текстов.

Исследование выполнено при поддержке РФФИ в рамках проектов № 18-00-01376 (18-00-00889) и № 18-00-01376 (18-00-00760).

2. Обзор существующих методов

Автоматическое выделение утверждений в исследуемом тексте. Разбиение текста на элементарные единицы проводится с использованием простых синтаксических моделей или синтаксического анализатора [12].

Разделение утверждений на аргументативные и неаргументативные. Для этого, как правило, применяются методы машинного обучения, которое реализуется на размеченных вручную текстах определенного жанра. Показано, что качество поиска существенно зависит,

прежде всего, от выбора признаков, а также от сочетания методов машинного обучения с методами на основе экспертных правил. Например, на этапе выявления предложений, содержащих аргумент, использование шаблонов с supervised probabilistic sequence model повысило F-меру на 17% [7].

Выявление роли утверждений в структуре аргументации. В работе [8] посылки и заключения выявляются методом SVM с F = 0.74, на этапе предварительной классификации утверждений на аргументативные и не содержащие аргумента работает метод максимальной энтропии. Классификационные признаки наряду со структурными, лексическими, синтаксическими, включают модель ближайшего контекста, дискурсивных и риторических отношений.

В работе [11] при выявлении утверждений (клауз) четырех типов (главный тезис, тезис, посылка, неаргументативное) лучшим из методов (SVM, наивный Байес, деревья решений и случайный лес) был признан SVM. Классификаторы тренировались на структурных, лексических (n-граммы (n = 1, 2, 3), глаголы, наречия, модальные слова), синтаксических, контекстуальных признаках, дискурсивных маркерах (объем словаря – 55 единиц).

Установление аргументативной связности утверждений. Иногда решение этой задачи предполагает и возможность вывода содержания одного предложения из другого, как в работе, проведенной несколькими группами исследователей [2] на парах фрагментов (предложений). Показано, что наилучшим образом такого рода связность может быть установлена со средней точностью 0.8 при использовании лексических, синтаксических признаков, меток семантических ролей, статистики n-граммного покрытия.

В работе [3] тематически сходные утверждения определяются с помощью модели на основе скрытого распределения Дирихле (LDA). Каждое слово характеризуется вероятностью принадлежности к некоторой теме. Расстояние между утверждением и его предшественником рассчитывается как евклидово расстояние между оценками по темам. Если расстояние ниже установленного порога, утверждение связывается с предыдущим. Если порог превышен, тогда вычисляется расстояние между утверждением и всеми предыдущими утверждениями и, если самое близкое из них находится на определенном расстоянии, связь устанавливается. Если ни один из этих критериев не соблюдается, предложение считается несвязанным ни с каким другим, встретившимся ранее. Доля совпавших с ручной разметкой связей равна 76,5 %. Направление связи не указывается.

Используя байесовский классификатор, который тренируется на посылках (premises) и заключениях (conclusion), авторы работы [4] идентифицируют отдельные компоненты схемы аргументации (рассмотрены две схемы – Expert Opinion and Positive Consequences) в тексте.

А затем с помощью дискурсивных маркеров определяют отношения атаки \ поддержки между утверждениями. Оставшиеся не присоединенными элементы схемы с помощью результатов тематического анализа текста LDA объединяются в структуру.

Следует отметить, что тематический анализ при установлении связей разного типа применяется все чаще. В настоящей работе для определения фрагментов текста, объединенных одной темой, используется профиль кластеризуемости слов и словосочетаний [1], с помощью которого можно установить потенциальные связи между утверждениями.

3. Постановка задачи

Предположим, что текст с помощью некоторого метода (например, одного из указанных в п. 2) разделен на утверждения $P = \{p_i\}$ (i – номер утверждения в тексте $i = 1, ..., M$; M – число утверждений в тексте), из которых извлечено подмножество аргументативных $\{p'_j\}$ ($j \le M$). Метод, описанный ниже в п.4, позволяет разбить текст на фрагменты $B^k = \{p_i\}^k$, содержащие утверждения, относящиеся к одной теме (k – номер фрагмента текста, $k = 1, ..., K$; K – число выделенных фрагментов). В данном исследовании нас интересует вклад тематического признака в эффективность распознавания аргументационной структуры. Будем считать, что множество $\{p'_j\}$ совпадает с множеством вручную выделенных аргументативных утверждений, а n_{re} – число связей, установленных вручную между утверждениями. Обозначим символами n_{rb} число потенциальных связей между утверждениями, которые выявляются по признаку принадлежности к одной теме. Пусть из них совпали с установленными вручную n_{ra} связей.

Задача состоит в получении оценок полноты (R) и точности (Pr) применения признака принадлежности утверждений (p'_{j1} и p'_{j2}) к одной теме B^k для установления связей между ними на основе вручную размеченных аргументативных утверждений и связей между ними: $R = n_{ra} / n_{re}$; $Pr = n_{ra} / n_{rb}$.

4. Построение профиля кластеризуемости

Построение тематической структуры текста в виде профиля кластеризуемости решается в предположении, что слова, неравномерно распределенные в тексте, являются тематически значимыми. Тогда задача построения тематической структуры может быть сведена к выявлению неравномерностей в позиционном распределении отдельных слов и словосочетаний в тексте согласно различным схемам расстановки точек на линии, если каждую точку рассматривать как место вхождения анализируемого слова (словосочетания) в

текст. Статистически значимые кластеры в этом случае успешно выявляются с помощью сканирующих статистик [9].

Пусть $x_1, x_2, \ldots, x_N$ – произвольный набор точек из единичного интервала (0, 1]. Требуется проверить гипотезу о равномерности (H_0) против альтернативы (H_1), связанной с тем или иным типом отклонения от равномерности. Для случая кластеризации эффективное решение основано на использовании сканирующей статистики $n(d)$, фиксирующей максимальное число точек n, попавших в интервал длины d при всевозможных расположениях этого интервала внутри единичного отрезка. Вычисление $n(d)$ ведется путем подсчета числа точек, попавших в окно ширины d, скользящее вдоль отрезка.

В данной работе вместо статистики $n(d)$ используется связанная с ней статистика $d(n)$, фиксирующая длину минимального интервала, содержащего ровно n точек ($2 \le n \le N$). Поскольку табулирование распределения этой статистики в широком диапазоне значений n и N представляется достаточно трудоемким, для оценки значимости отклонения вычисленной на конкретном тексте статистики $d(n)$ от значения, постулируемого гипотезой H_0 (равномерность), можно прибегнуть к имитационному моделированию.

Схема выявления позиционных аномалий (типа кластеризации) в распределении лексических единиц выглядит следующим образом (см. [1]).

1. В нормализованном тексте подсчитывается частота встречаемости каждой леммы.

2. Пусть x – произвольная лемма (или лемматизированное словосочетание), $F(x)$ – суммарное число ее (его) вхождений в текст, n – фиксированное число последовательных вхождений x в текст ($2 \le n \le F(x)$), $d(n)$ – длина минимального фрагмента текста, содержащего n вхождений x. Для дальнейшего анализа отбираем леммы x со значением $F(x) \ge F_b$, где F_b – пороговое значение частоты, зависящее от длины текста N в словоформах.

3. Для каждой леммы проводим перебор по всем допустимым значениям n ($F_b \le n \le F(x)$). Для фиксированного n:

а) вычисляем значение $d(n)$ в анализируемом тексте;

б) с помощью имитационного моделирования оцениваем распределение этой статистики при гипотезе H_0. Для этого путем многократного перемешивания слов в исходном тексте формируем m его рандомизированных аналогов с равномерным распределением слова x по тексту (приемлемыми считаются значения $m \ge 100$). По полученной подборке вычисляем оценки минимального и среднего значения статистики $d(n)$ (соответственно, $S_{\min}$, S_{avr}), а также среднеквадратичное отклонение s.

4. Сравниваем наблюдаемое в исходном тексте значение статистики $d(n) = S_o$ с оценками, полученными в имитационном эксперименте. Считаем, что аномальное (неслучайное) отклонение от равномерности типа «кластеризация» имеет место, если выполняется условие: $(S_o \leq S_{\min})$ and $(S_o \leq S_{avr} - 3s)$.

Значимость выделенного кластера можно характеризовать безразмерной величиной $\delta(x)$, равной отношению среднего расстояния между вхождениями x к среднему внутрикластерному расстоянию между вхождениями x.

Понятие *профиля кластеризуемости* вводится для того, чтобы аккумулировать на одном графике информацию обо всех участках кластеризации разных лемм. Формально, профиль кластеризуемости – это ступенчатая функция, аргументом которой является порядковый номер предложения в тексте, а значение равно числу различных кластеров, включающих в себя данное предложение. При этом в отдельно рассматриваемом предложении могут не присутствовать одновременно все леммы, кластеризующиеся в данном участке текста. Пики профиля кластеризуемости обычно соответствуют отдельным темам текста, а провалы между ними – переходу от одной темы к другой. В профиле отражаются различные связи (часть\целое, общее\частное, ассоциации) между леммами, кластеризующимися в одном и том же участке текста.

Ниже приведен пример профиля кластеризуемости лемм научно-популярного текста «Люди, которые создают нормы» (см. Приложение), рассказывающего о том, как могут и должны формироваться языковые нормы ($\delta(x) = 2$).

n_s	n_c	*lemma*
1	1	РУССКИЙ ЯЗЫК;
7	3	НОРМА; ВАРИАТИВНОСТЬ; РУССКИЙ ЯЗЫК;
8	2	НОРМА; РУССКИЙ ЯЗЫК;
10	1	РУССКИЙ ЯЗЫК;
12	2	ЯЗЫК; РУССКИЙ ЯЗЫК;
17	1	ЯЗЫК;
18	2	ЯЗЫК; ИЗМЕНЕНИЕ;
19	1	ИЗМЕНЕНИЕ;
24	2	ЯЗЫК; ИЗМЕНЕНИЕ;
27	1	ИЗМЕНЕНИЕ;
30	2	ИЗМЕНЕНИЕ; ПРОСТРАНСТВО;
34	1	ПРОСТРАНСТВО;
37	3	ЯЗЫК; ЗАДАЧА; ПРОСТРАНСТВО;
38	4	ЯЗЫК; ЗАДАЧА; ПРОСТРАНСТВО; НОРМА;
40	3	ЗАДАЧА; ПРОСТРАНСТВО; НОРМА;
43	1	ЗАДАЧА;
46	3	ЗАДАЧА; ЭТАП; ТЕЛЕВИДЕНИЕ;
47	4	ЗАДАЧА; РЕЧЬ; ЭТАП; ТЕЛЕВИДЕНИЕ;
51	2	ЗАДАЧА; РЕЧЬ;
53	1	РЕЧЬ;

Рис.1. Профиль кластеризуемости текста «Люди, которые создают нормы»

Ось абсцисс с номерами предложений направлена вниз, а ось ординат (число кластеров) – по горизонтали: слева направо. Для экономии места ось ординат представлена в нелинейном масштабе: указаны номера (n_s) лишь тех предложений, на которых происходит изменение значений профиля, т.е. добавляются новые кластеры или исчезают старые (n_c – число кластеризующихся лемм). Наибольшее значение профиля равно 4 и зафиксировано в предложениях под номерами 38 и 47. Профиль на Рис.1 демонстрирует наличие нескольких явно выраженных тем в тексте, например, в диапазоне (1–10), (34–43), (43–53). Темы в диапазоне (10–34) являются менее выраженными и могут быть присоединены к смежным.

Рассматриваются возможные взаимные расположения кластеров – вложенность, пересечение, разнесенность, относительно которых сформирована гипотеза: связанные аргументативные утверждения могут находиться в тех фрагментах текста, которые входят в кластеры одной темы или в пересекающиеся по лексическому составу кластеры смежных тем. Вероятность существования связи аргументативных утверждений, принадлежащих фрагментам несмежных тем, существует, но она существенно ниже и ее установление требует введения дополнительных правил.

5. Результаты эксперимента

Материалом для эксперимента по оценке эффективности признака принадлежности \ непринадлежности связанных аргументативных утверждений к одной теме текста послужили 100 размеченных вручную текстов, длиной не менее 500 слов.

Для описания структур аргументации принят стандарт, фиксированный форматом AIF (Argument Interchange Format) [10]. Согласно данному формату аргументы представляются ориентированным графом, в котором выделяют два типа вершин: информационные вершины (вершины-утверждения) и вершины-схемы (вершины-аргументы). Вершинам-утверждениям сопоставляются посылки, заключения, а вершинам-схемам – типовые схемы (модели) рассуждений. Для разметки предлагались к использованию несколько десятков схем аргументации из компендиума Уолтона [13].

Всего 36 разных схем аргументации были употреблены экспертами при разметке текстов коллекции более 2-х раз. Наиболее частотные из них (с частотой встречаемости $F > 10$) представлены в таблице 1 ниже. Как можно видеть, примеры Example_Inference в научно-популярных текстах являются самым распространенным видом аргументации, их доля от всех реализованных в размеченной коллекции составляет ~ 17 %. Отстает по частоте, но является весьма характерным для научно-популярного дискурса обращение к аргументации, основанной на строгом (неоспоримом) и нестрогом причинно-следственном выводе

(ModusPonens_Inference и CauseToEffect_Inference). Стилистически значимое обращение к мнению экспертов в обсуждаемой области (ExpertOpinion_Inference) является сильным аргументом для убеждения аудитории как научно-популярного, так и научного текста. Наконец, еще одним значимым вариантом аргументации с помощью каузальных отношений является абдуктивный вывод, позволяющий осуществлять принятие объяснительных правдоподобных гипотез, как в случае аргументации по схеме Sign_Inference. Доля пяти редких схем (CausalSlipperySlope_Inference, DirectAdHominem_Inference, Gradualism_Inference, InconsistentCommitment_Inference, PropertyNotExistant_Conflict) с $F = 2$ составила в сумме 0,3 % от всех использованных в разметке коллекции.

Таблица 1. Частоты применения схем аргументации в научно-популярных текстах

№ п/п	Название схемы	*F*	№ п/п	Название схемы	*F*
1	Example_Inference	1008	12	PopularOpinion_Inference	70
2	CauseToEffect_Inference	686	13	VerbalClassification_Inference	50
3	ModusPonens_Inference	632	14	ExceptionalCase_Inference	38
4	ExpertOpinion_Inference	406	15	Logical_Conflict	30
5	Sign_Inference	308	16	NegativeConsequences_Inference	28
6	CorrelationToCause_Inference	228	17	PositiveConsequences_Inference	26
7	EstablishedRule_Inference	100	18	Bias_Inference	16
8	PracticalReasoning_Inference	94	19	PopularPractice_Inference	14
9	EvidenceToHypothesis_Inference	82	20	Waste_Inference	14
10	PositionToKnow_Inference	80	21	Commitment_Inference	14
11	Analogy_Inference	78			

Рассмотрим пример аргументативной разметки текста (Рис. 2). Текст полностью приведен в приложении. Номера предложений в тексте соответствуют указанным в профиле кластеризуемости (Рис. 1). Синим в тексте и в профиле выделен фрагмент, для которого на рисунке 2 приводится структура аргументации. В прямоугольники помещены аргументативные утверждения (в тексте пронумерованы с префиксом S и выделены квадратными скобками), в овалах указаны названия схем аргументации. Утверждение на сером фоне в тексте не присутствует явно, но вводится экспертом, который делает разметку, для устранения лакун в аргументации. Утверждения S23 и S24, а также S24 и S32 связаны, т.к. принадлежат одной теме. Утверждения S32 и S37, S32 и S27, S23 и S25, S24 и S25, S25 и S27 принадлежат смежным пересекающимся темам. Проблемы с установлением связи

возникают с позиционно разнесенными утверждениями (S37 и S2). Но в приводимом примере кластер, который формируется во фрагменте текста, содержащем утверждение S2, образован словосочетанием РУССКИЙ ЯЗЫК, а тема, к которой принадлежит утверждение S37 содержит слово ЯЗЫК. Если принять во внимание, что эти лексические единицы связаны отношением общее-частное, то можно предположить, что и утверждения связаны.

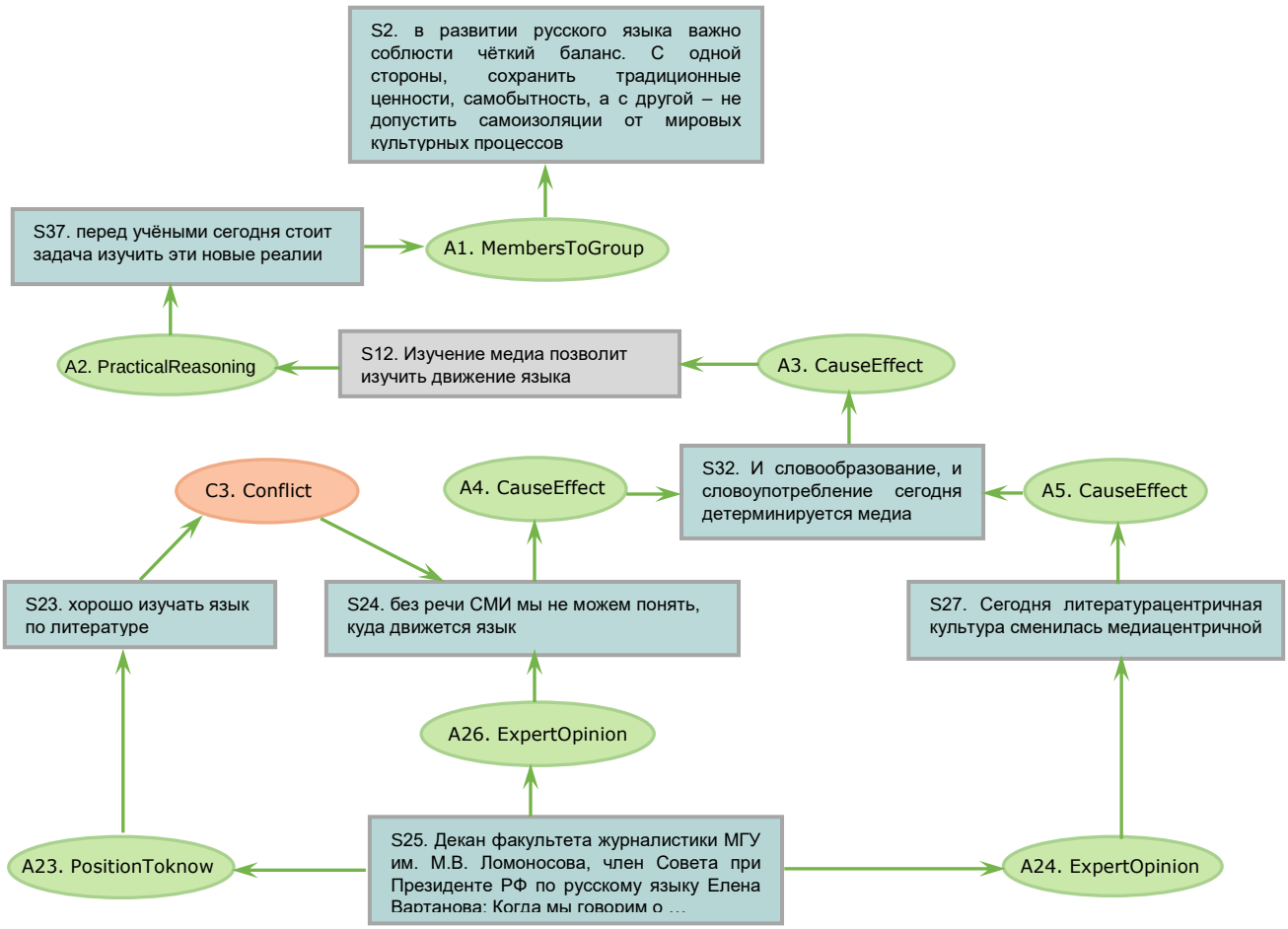


Рис. 2. Фрагмент аргументационной структуры текста «Люди, которые создают нормы»

Если схема аргументации реализуется в одном предложении (доля таких случаев от всех зафиксированных составляет около 15 %), то нет необходимости использовать признак принадлежности к одной теме или смежным пересекающимся темам. В других случаях проверка тематического признака может быть полезна. В частности, для всех рассмотренных схем вычисленная с макро-усреднением полнота (R) установления связей между утверждениями составила 76,2 %, а точность (Pr) – 48,6 %. Явно непредставленные в тексте утверждения не учитывались при подсчете полноты и точности.

Соответствующая разным типам схем рассуждений (представлены 5 самых частотных) полнота установления связей указана в таблице 2. Распознавание этих схем могло бы

послужить основой для автоматического восстановления цепочек аргументов с целью получения максимально полной структуры аргументации.

Таблица 2. Полнота выявления связанных утверждений для разных схем рассуждений

№ п/п	Название схемы	*R*
1	Example _Inference	78%
2	CauseToEffect _Inference	86%
3	ModusPonens _Inference	77%
4	ExpertOpinion _Inference	72%
5	Sign _Inference	82%

Ошибки в установлении связей случаются из-за наличия в аргументационной структуре большого числа утверждений, явно непредставленных в тексте. Также связи не определяются, если утверждения находятся вне зоны кластеризации. Преимущественно такая ситуация характерна для утверждений, встречающихся в самом начале текста. Трудность в установлении связи представляют собой и позиционно разнесенные утверждения, принадлежащие разным темам. Пересечение множества терминов, характеризующих такие темы, часто отсутствует.

Сравнение точности и полноты распознавания связей, полученных в других работах, возможно только по результатам экспериментов, проведенных на материале английского языка. Но условия экспериментов слишком различаются, сравнение вряд ли можно считать правомерным. Самыми близкими являются результаты работы [3]. Полнота и точность (76,5 % и 72,2 % соответственно) вычислена на очень ограниченном наборе – всего 26 утверждений. В нашей работе получена сравнимая оценка по полноте (76,2 %) и уступающая по точности (48,6 %).

Возможности улучшения метода связаны с самыми разными аспектами автоматической обработки текста. К ним относятся: разрешение анафоры; отбор сочетаний слов определенного типа; изменение параметра, отвечающего за плотность кластера (его уменьшение приводит к образованию более слабых кластеров, но позволяет выявить недостающие связи); учет общих слов и словосочетаний в разных темах; учет конфигурации уже построенной части аргументационной структуры.

Существуют и неустранимые ограничения, чаще всего связанные с недостаточным объемом анализируемого текста, когда либо кластеры не обнаруживаются, либо весь текст попадает в один кластер или одну тему.

6. Заключение

В статье приведены результаты исследования признака принадлежности утверждений к одной теме для установления связи между ними. Тематическая структура строилась автоматически на основе позиционного распределения слов и словосочетаний, выявляемого с помощью сканирующих статистик. Для верификации связей, извлекаемых из тематической структуры, использована коллекция размеченных текстов. Рассматривалась возможность автоматического установления связи между утверждениями, выделенными экспертами в ходе разметки коллекции. Полученные оценки полноты сравнимы с известными, точность несколько уступает. Однако потенциал метода нельзя считать исчерпанным. Исследования по его применению в обнаружении связи между утверждениями будут продолжены.

Список литературы

1. Гусев В.Д., Мирошниченко Л.А., Саломатина Н.В. Тематический анализ и квазиреферирование текста с использованием сканирующих статистик // [Электронный ресурс]. URL: http://www.dialog-21.ru/media/2374/gusevvd.pdf (дата обращения: 27.08.2020).
2. Dagan I., Glickman O., and Magnini B. The PASCAL recognising textual entailment challenge // Machine Learning Challenges, Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Textual Entailment, ed. Quinonero-Candela J. and others, Springer, 2006. P. 177–190.
3. Lawrence J., Reed C. and others. Mining arguments from 19th century philosophical texts using topic based modelling. // Proc. of the First Workshop on Argumentation Mining, Baltimore, Maryland, June. Association for Computational Linguistics (ACL), 2014. P. 79–87.
4. Lawrence J., Reed C. Combining Argument Mining Techniques // Proc. of the 2nd Workshop on Argumentation Mining (Denver), 2015. Vol. 1. P. 127–136.
5. Lawrence J, Reed C. Argument Mining: A Survey // Computational Linguistics, 2019. Vol. 45. N 4. P.765–818.
6. Lippi M., Torrony P. Argumentation Mining: State of the Art and Emerging Trends // ACM Transactions on Internet Technology, 2016. Vol. 16. Article 10.
7. Madnani N., Heilman M., Tetreault J., Chodorow M. Identifying high-level organizational elements in argumentative discourse // Proc. of the 2012 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Montreal), 2012. P. 20–28.
8. Mochales Palau R., Moens M-F. Argumentation mining: The detection, classification and structure of arguments in text // Proc. of the 12-th Int. Conf. on Artificial Intelligence and Law (Barcelona). ACM, New York, 2009. P. 98–109.

9. Naus J. The distribution of the size of the maximum cluster of points on a line // Journal of the American Statistical Association. Electronic materials, 1965. Vol. 60. P. 532–538.
10. Rahwan I., Reed C. The argument interchange format // Argumentation in artificial intelligence, ed. Rahwan I. and Simari G. Springer, 2009. P. 383–402.
11. Stab C., Gurevych I. Identifying argumentative discourse structures in persuasive essays // Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing (Doha), 2014. P. 46–56.
12. UDpipe. URL: http://github.com/ufal/udpipe (дата обращения: 25.09.2020).
13. Walton D., Reed C., Macagno F. // Argumentation schemes Fundamentals of critical argumentation. New York: Cambridge University Press, 2008. 443 p.

Приложение

Текст статьи «Люди, которые создают нормы» (Авторы: М. Осадчий, Е. Вартанова, Т. Горяева.)

1. 19 мая состоялась встреча Президента Владимира Путина с представителями Совета по межнациональным отношениям и Совета по русскому языку.

2. Президент в ходе общения отметил, что [S2. в развитии русского языка важно соблюсти чёткий баланс.

3. С одной стороны, сохранить традиционные ценности, самобытность, а с другой – не допустить самоизоляции от мировых культурных процессов.]

4. На пресс-конференции, приуроченной ко дню русского языка 6 июня, эксперты обсудили, по каким законам развивается язык, роль СМИ в его изменении и как выработать ответственное отношение у носителей к существующим трансформациям.

5. Проректор по науке Государственного института русского языка им. А.С. Пушкина Михаил Осадчий:

6. Речевая коммуникация – это явление очень динамичное.

7. Мы всегда имеем дело с некоей вариативностью: вариативностью норм, вариативностью реализации тех или иных компетенций и так далее.

8. Сегодня не утихают споры по поводу существования нормы как таковой.

9. Когда мы говорим о нормах в языке, мы в первую очередь имеем в виду словарь.

10. Словарь составлен такими же людьми, причём, как правило, по результатам опросов.

11. То есть, идёт наблюдение за узусом, реальной речевой коммуникацией, на основе которой уже закрепляются те или иные правила.

12. Эти тренды в некоторой степени входят в противоречие с идеей государственного нормирования языка.

13. С одной стороны, русский язык – это язык государства, и тут он совершенно точно должен нормироваться и кодифицироваться.

14. С другой стороны, русский – это язык общества, инструмент межличностного и межнационального взаимодействия.

15. И в этом плане многообразие языка, не выходящее за рамки приличия, должно поддерживаться, развиваться и изучаться.
16. Необходимо поддерживать проекты, связанные со сбором, анализом и популяризацией различной литературы на русском языке, принимая во внимание, что литература является высшей формой функционирования языка.
17. Если носитель, изучая литературу различных эпох, видит, как развивается язык, он по-другому воспринимает саму идею изменчивости языка.
18. Сегодня я часто сталкиваюсь в школе с тем, что учителя однозначно негативно относятся к любым изменениям в языке, будь то произношение, грамматика или общие нормы коммуникации.
19. Но нужно приучать и преподавателей, и учеников к тому, что эти изменения неизбежны.
20. Для этого необходимо вырабатывать некое осмысленное и ответственное отношение к изменениям.
21. В менеджменте есть такой важный элемент любого управленческого процесса – менеджмент изменений.
22. Мы, носители языка, тоже являемся своего рода менеджерами своего языка (кто нам может приказать, как говорить).
23. Но навыками управления изменениями надо овладевать уже сейчас, чтобы понимать, какие модификации мы принимаем, а какие, мотивированно и осознанно, отвергаем.
24. В любом случае цель государства и образования – формирования у своего населения элитарного отношения к языку и элитарных навыков пользования этим языком.
25. [S25. Декан факультета журналистики МГУ им. М.В. Ломоносова, член Совета при Президенте РФ по русскому языку Елена Вартанова:
26. Когда мы говорим о распространении новых норм, всех, конечно, волнует речь журналистов.]
27. Здесь я принимаю на себя все удары.
28. Конечно, журналисты могут говорить неправильно, но я бы хотела выступить в защиту медиаречи.
29. Думаю, что [S23. хорошо изучать язык по литературе.]
30. Так было, особенно в XIX веке, когда литература была главным культурным, интеллектуальным и даже академическим пространством.
31. Но сегодня, в эпоху, когда случилась цифровая революция, когда интернет и маленькие гаджеты проникли в самое интимное пространство человеческого бытия, [S24. без речи СМИ мы не можем понять, куда движется язык.]
32. [S32. И словообразование, и словоупотребление сегодня детерминируется медиа.]
33. При этом, даже если что-то в этом движении нам не нравится, присмотреться к изменениям, которые приживаются в языке после использования журналистами, кажется целесообразным.
34. [S27. Сегодня литературацентричная культура сменилась медиацентричной.]
35. Если раньше в центре был текст, то сейчас его место заняла картинка.

36. Но картинка не обедняет речь, она просто приводит звучащую речь, которая становится важным фактором влияния на речевую среду.

37. Поэтому [S37. перед учёными сегодня стоит задача изучить эти новые реалии], а перед журналистами стоит задача повышения квалификации в области русского языка.

38. К тому, что должно детерминировать норму – правила, разговорный язык или, может быть, язык журналиста – может быть разное отношение.

39. Но очевидно одно: если норма не будет учитывать движение языка, даже в самых просторечных его проявлениях, тогда она потеряет свою легитимность для значительного числа людей.

40. Тут важно прислушаться и к таким новым участникам медиапространства, как блогеры.

41. Закон о блогерах, принятый около года назад, может быть несовершенен, но одно он показывает точно: эти люди становятся ориентиром для многих, а значит, должны нести определённую ответственность.

42. С другой стороны, понятно, что без нормы мы можем оказаться в пространстве социального непонимания, потому что речевая коммуникация является основой социальной коммуникации, и здесь нужны правила игры.

43. Сегодня учёные начинают говорить о лингвоэтике как о важнейшем критерии оценки речи в межнациональном государстве, когда словоупотребление может стать причиной конфликтов.

44. На совместном заседании Советов Владимир Путин признал, что важно, чтобы государственные служащие тоже хорошо владели языком и периодически проходили курсы повышения квалификации наравне с работниками СМИ.

45. Директор Российского государственного архива литературы и искусства, член Совета при Президенте РФ по русскому языку Татьяна Горяева: Мы выступаем как фиксаторы нормы.

46. Интересно, что и радиовещание, и телевидение сегодня вернулись к своему первоначальному этапу – к прямому эфиру.

47. Известно, что на пионерском этапе развития телевидения в основном передавались прямые трансляции, и естественно, речь там звучала живая и не всегда корректная.

48. Потом наступила эпоха звукозаписи.

49. Тому поспособствовали разные обстоятельства, в том числе, развитие технических средств записи.

50. Тогда телевидение и радио перешли на этап фиксирования.

51. И уже на наших глазах расцвело непосредственное вещание, когда мы услышали живую речь.

52. И задача здесь – сохранить баланс между непосредственной речью журналиста и соблюдением им норм литературного языка.

53. Что касается цензуры и проверки не только содержания, но и формы подачи информации, сегодня скорее речь идёт о самоцензуре.

54. Ведь внешние институты – Главлиты, Главреперткомы и т.д. – сегодня не функционируют, после знаменитого постановления 1969 года о передаче власти Главлита фактически в руки редакторов СМИ, журналов и газет.

55. С этого момента ответственность за содержание полностью ложилась на плечи главного редактора.

56. Сейчас есть определённые факторы, которые напоминают эти времена.