DOI: 10.26565/2304-6201-2020-46-07

УДК 004.2

Analysis of existing verification technologies for parallel programs

O.Yu. Moroz, O.G. Tolstoluzka, R.V. Savchenko

Moroz Olha Yuriivna — *Senior lecturer of the Department of Theoretical and Applied Systems Engineering, Faculty of Computer Science; VN Karazin Kharkiv National University, 6 Svobody Square, Kharkiv, Ukraine, 61022*
e-mail: o.moroz@karazin.ua
https://orcid.org/0000-0002-4920-4093.

Tolstoluzka Olena Gennadiivna — *Doctor of Technical Sciences, Senior Research Fellow, Professor of the Department of Theoretical and Applied Systems Engineering, Faculty of Computer Science; VN Karazin Kharkiv National University, 6 Svobody Square, Kharkiv, Ukraine, 61022*
e-mail: elena.tolstoluzka@karazin.ua
https://orcid.org/0000-0001-2741- 180.

Savchenko Roman Valeriyovich — *student of the Faculty of Computer Science; VN Karazin Kharkiv National University, 6 Svobody Square, Kharkiv, Ukraine, 61022*
e-mail: roma.savchenko@gmail.com
https://orcid.org/ 0000-0002-8184-1818

There have been large fluctuations in the perceived value of parallel computing in the last decades. Sometimes parallel computation has been viewed optimistically as the solution to all of our computational limitations. The conventional division of verification methods is analyzed. It has been concluded that synthetic methods of software verification can be considered as the most relevant, useful and productive ones. The relevance of the implementation of the methods of formal verification of software of computer systems, which supplement the traditional methods of testing and debugging, and make it possible to improve the program uptime and security, is noted. The formal verification methods of computer system software can guarantee that verification of the properties is performed by a system model. Nowadays, development of these methods is lying in the direction of reducing the formal verification total cost, support of modern programming concepts and minimization of "manual" work in the transition from the system model to its implementation. Their main feature is the ability to search for errors using a mathematical model without recourse to the existing software realization. It is very convenient and economical. There are several specific techniques used for a formal model analysis, such as a deductive analysis, model and consistence check. Every verification method is being used in particular cases, depending on the goal. Synthetic methods of software verification are considered as the most actual, useful and efficient, as they aim to combine the advantages of different verification approaches being free of their drawbacks. The significant progress in the development of such methods and their implementation in the industrial software development has been currently made.

Keywords: *parallel programming; verification of programs; formal verification*

Аналіз існуючих технологій верифікації паралельних програм

Мороз Ольга Юріївна — *старший викладач кафедри теоретичної та прикладної системотехніки факультету комп'ютерних наук; Харківський національний університет імені В. Н. Каразіна, майдан Свободи, 6, Харків, Україна, 61077*

Толстолузька Олена Геннадіївна — *д.т.н., с.н.с., професор кафедри теоретичної і прикладної системотехніки факультету комп'ютерних наук; Харківський національний університет імені В. Н. Каразіна, майдан Свободи, 6, Харків, Україна, 61077*

Савченко Роман Валерійович — *студент факультету комп'ютерних наук; Харківський національний університет імені В. Н. Каразіна, майдан Свободи, 6, Харків, Україна, 61077*

Проведено аналіз загальноприйнятого поділу методів верифікації. Суть формальних методів полягає в створенні математичних моделей програм і вимог і в логічному аналізі відповідності між побудованими моделями. На сьогодні, формальні методи ¬ це фундамент, на якому стоїть будівля програмної інженерії. Зроблено висновок, що найактуальнішими, найбільш корисними та продуктивними можна вважати синтетичні методи верифікації ПЗ. Зазначено, що актуальним є впровадження в практику методів формальної верифікації програмного забезпечення комп'ютерних систем, що доповнюють традиційні методи тестування і налагодження, і дозволяють підвищити безвідмовність і безпеку програм.
Ключові слова: *паралельне програмування; верифікація програм; формальна верифікація*

Анализ существующих технологий верификации параллельных программ

Мороз Ольга Юрьевна — *старший преподаватель кафедры теоретической и прикладной системотехники факультета компьютерных наук; Харьковский национальный университет имени В. Н. Каразина, площадь Свободы, 6, Харьков, Украина, 61077*

Толстолужская Елена Геннадиевна — *д.т.н.,* с.н.с., *профессор теоретической и прикладной системотехники факультета компьютерных наук; Харьковский национальный университет имени В. Н. Каразина, площадь Свободы, 6, Харьков, Украина, 61077*

Савченко Роман Валерьевич — *студент факультета компьютерных наук; Харьковский национальный университет имени В. Н. Каразина, площадь Свободы, 6, Харьков, Украина, 61077*

Проведен анализ общепринятого распределения методов верификации. Суть формальных методов состоит в создании математических моделей программ и требований и в логическом анализе соответствия между построенными моделями. На сегодня, формальные методы ¬ это фундамент, на котором стоит здание программной инженерии. Сделан вывод, что наиболее актуальными, наиболее полезными и продуктивными можно считать синтетические методы верификации ПО. Отмечено, что актуальным является внедрение в практику методов формальной верификации программного обеспечения компьютерных систем, дополняющие традиционные методы тестирования и отладки, и позволяют повысить безотказность и безопасность приложений.

Ключевые слова: параллельное программирование; верификация программ; формальная верификация

There have been large fluctuations in the perceived value of parallel computing in the last decades. Sometimes parallel computation has been viewed optimistically as the solution to all of our computational limitations. But there are those who argue that it is not worth the effort, given that the CPU speed continues to improve and memory cost to reduce. The perceptions tend to fluctuate between these two extremities due to a number of factors: constant changes in the “hot” issues to be addressed, availability to users of the programming environment, the computer market, vendors involved in building the supercomputers and the focus of the scientific community at any given moment and time. As a result, it is very difficult to judge objectively the value and prospects of parallel computing.

Verification of programs or computer systems is an activity aimed at determining their correctness or finding errors. This activity can take various forms: code inspection; static analysis (search for typical errors); testing (running the program on examples and checking the correctness of the results); simulation modeling (creation of a feasible model of the system and its inspection in a special environment); formal verification (construction of a logical model of the system and its analysis by means of mathematical logic). There are many approaches, but none of them can guarantee the correctness of really complex projects; the best results, as practice shows, are achieved by combining different methods. The essence of formal methods is to create mathematical models of programs and requirements and a logical analysis of the correspondence between the constructed models. Today, formal methods are the foundation on which the building of the software engineering stands. It should be noted that formal methods have long gone beyond the academic community and have become part of the industry, as well as part of the system development process (microprocessors and operating systems). In the information society, software development has become mass activity.

Software Testing is developing both as industry and science. Distributions occur, new directions and scientific flows appear; different techniques, methods and practices of software testing are used. It is often caused by the fact that companies or organizations pursue different goals or by specifics of testing the different product categories (medicine, tourism, education, finance, e-commerce, etc.).

STLC (Software Testing life cycle) means all actions which are performed during testing of software product.

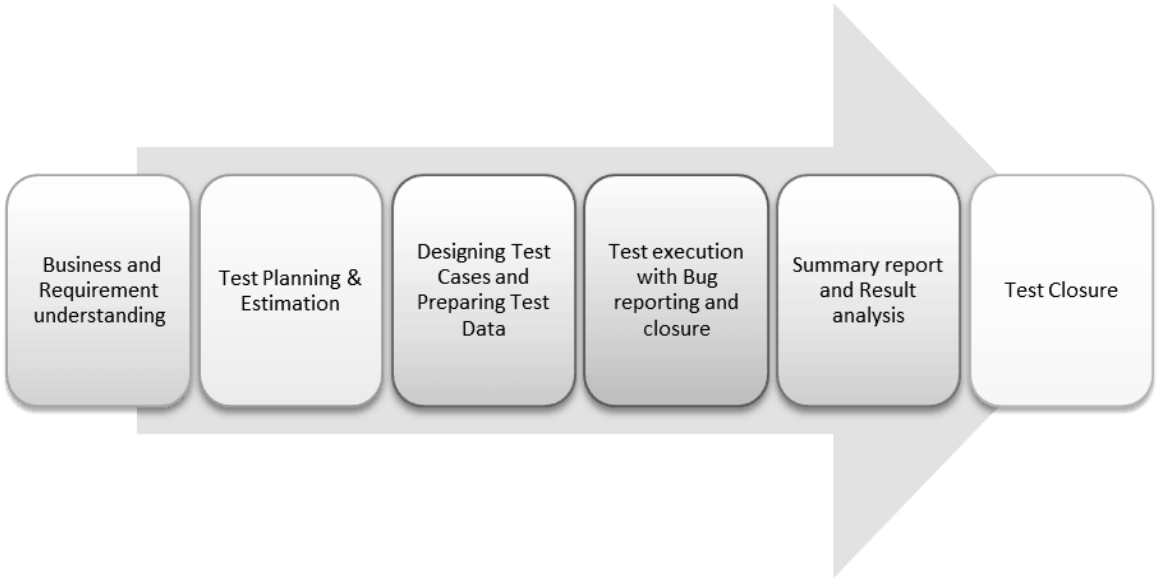

Figure 1. STLC (Software Testing life cycle)

Nowadays, ensuring high reliability and infallibility of modern management systems software is one of the main tasks of the IT industry. Therefore, the implementation of software engineering formal verification methods, which complement traditional testing and tune-up methods, and allow increasing software infallibility and security is highly relevant.

Software verification is more general concept than testing. The main goal of verification is to achieve a guarantee that a verified object (requirements or program code) meets the demands, is implemented without unexpected functions and suits design specifications and standards. The common division of verification methods is presented as a diagram in Fig. 2.

Software system validation is a process of proving that goals set for this system are achieved as a result of system development. In other words, validation is checking if the system meets the customer's expectations.

The main standard, which controls software verification planning and realization, is called IEEE 1012 for verification and validation processes [1]. This standard includes the description of different verification task sets which accord to diverse activities, recommended template of the inspection and confirmation plan, determination of 4 software criticality levels (from high to minimal).

Software expertise includes all verification methods, in which evaluation of the software life cycle artifacts is done by people. The main advantage of this method is the ability to detect 50-90% of errors [5]. However, it also has some drawbacks. Error search, evaluation and analysis of software properties are done by a human (usually it is a group of 2-5 persons). Therefore, experts, programmers with 10 or more years of work experience are required to perform this action.

Static analysis is an analysis performed without program execution. Its methods can be divided into two: the control to ensure that all formalized rules of correct construction of these artifacts are performed, and the search for some typical errors and defects based on some templates. [2]. Static analysis instruments often use both kinds of checking. Static analysis is considered to be the most often used verification method. Proven code correctness rules or common errors templates are being transferred to the development environments.

Static analysis advantages:

– Automatic analysis of multiple execution paths at one time.
– Detection of errors that occur only on single execution paths or on unusual input data.
– Ability to analyze on an incomplete set of input files.
– No overhead costs during program execution.

Drawbacks of this method:

– Lot of error responses.
– Manual checking of work results is required (needs significant time, human and material resources).

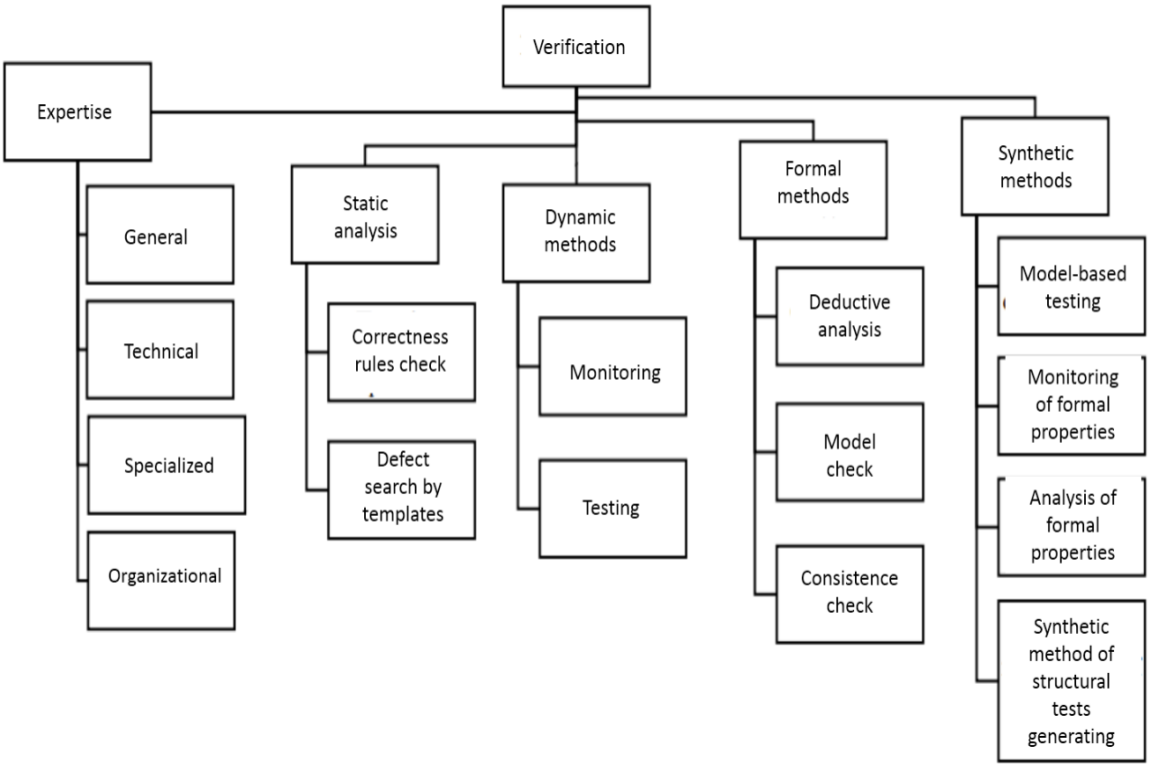

Figure 2. Common division of verification methods

Dynamic verification methods use results of real work of software system or its prototypes to verify the compliance of these results with the requirements and designs.

There are two main kinds of dynamic verification methods: monitoring, which provides only observation, recording and evaluation of the software normal working process, and testing, which runs software by using prepared scripts. The advantage of this method is good error detecting ability; the drawbacks include the need for a prepared input data set, runtime environment, as well as high demand for the resources.

Formal verification methods. Their main feature is an ability to search for errors by using mathematical model without recourse to existing realization of software. It is very convenient and economical. There are several specific techniques used for formal models analysis, such as a deductive analysis, a model and a consistence check. Unfortunately, construction of these models requires a correct and adequate model of software itself [4]. Only if model is created correctly, some of its properties could be analyzed automatically. However, in most cases specialists require deep knowledge of mathematical logic, algebra and must be experienced users of this device in order to do an effective analysis.

Compared to sequential writing, parallel programs are much more complicated. In sequential programming, a programmer develops an algorithm, then expresses it to the computer in a correct and understandable language, and makes it efficient to execute. Parallel programming has the same problems, but also a number of additional ones. They complicate the development and have no analogues in the consistent sphere. These problems include: data allocation management, parallelism

search and expression, computational load, as well as the proper implementation of a parallel algorithm, interprocessor communication management, and balancing. The algorithm takes turns considering each of these challenges.

Distribution of task data is another problem of parallel programming. Most conventional parallel computers have the concept of data locality. This means that you can get much faster access to data stored in memory that is “closer” to a particular processor. Data locality can also occur because each processor has its own local memory. Similarly, in a machine with allocated memory, this occurs through a cache processor that is in the shared memory system.

The programmer must pay attention to where the data is stored in relation to the processors that will access them, precisely because of the influence of data localization. The closer the data, the faster the processor will be able to access them and shut down. That is why the division of labor and the dissemination of data are closely linked. The optimal design of the program will take both aspects into account.

For a long time, the basis of parallel computing has been the automatic parallelization of a serial program by the compiler. Decades of work by compiler researchers have had little success, and automatic parallelization works only in limited conditions. In the best case, the languages are partially implicit, in which case a programmer gives the directives for parallelizing the compiler. Verilog, VHDL, Parallel Haskell, SISAL, Mitrion-C, and System C (for FPGA) are known as implicit parallel programming languages.

The tools for automatically generating code-based tests that use additional sources of information have been recently developed actively.

These sources include static a code analysis, a formal analysis, earlier test monitoring, etc. Since this kind of instruments uses 3–4 different techniques, their methods are classified as a separate kind of synthetic verification methods [3]. They combine several approach types – static analysis, formal analysis of software properties and testing. In the last 10–15 years some of those methods such as primarily model-based testing and monitoring of formal properties have become independent research areas. Advantages and drawbacks of synthetic methods are determined by combination of verification methods used by a synthetic method.

The purpose of the work is to analyze the possibilities of using structures of the semantic-numerical specification to describe formally the objects of the model of planning parallel processes in the computer networks. This work contains the analysis of the existing methods of static and time model specification. The generalized model planning parallel processes in the computer networks is analyzed.

The formal description of input data of the planning model by the semantic-numerical specification is proposed. The implementation program of graphic editor for the graphic specification for the model planning parallel processes objects has been developed. The analysis of the resource model planning highlights its significance and necessity of an object format. With the help of the program we can construct a topology of the computer networks, C-graphs for C-programs, in order to understand and create structures of the semantic-numerical specification for input data of the planning model.

The formal verification methods of computer system software can guarantee that verification of the properties is performed by a system model. Nowadays, development of these methods is lying in the direction of reducing the formal verification total cost, support of modern programming concepts and minimization of "manual" work in the transition from the system model to its implementation.

Every verification method is been used in particular cases, depending on the goal. Every verification method is being used in particular cases, depending on the goal. Synthetic methods of software verification are considered as the most actual, useful and efficient, as they aim to combine the advantages of different verification approaches being free of their drawbacks.

The significant progress in the development of such methods and their implementation in the industrial software development has been currently made.

ЛІТЕРАТУРА

1. IEEE 1012-2004 Standard for Software Verification and Validation. IEEE, 2005. p.153.
2. L. Yu A light-weight static approach to analyzing UML behavioral properties. L. Yu, R. B. France, I. Ray,K. Lano.. Proc. of 12-th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007) pp. 56–63, 2007. p. 79.

https://www.cs.colostate.edu/~iray/research/papers/iceccs07.pdf

3. M. Broy Model Based Testing of Reactive Systems. M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, A.Pretschner (eds.). LNCS 3472, Springer, 2005. p. 273.
4. T. Ball Thorough Static Analysis of Device Drivers. In Proc. of EuroSys 2006. T. Ball, E. Bounimova, B.Cook, V. Levin, J. Lichtenberg, C. McGarvey, B. Ondrusek, S. K. Rajamani, A. Ustuner., ACM SIGOPS OperatingSystems Review, 2006. p. 74.
5. Y. K. Wong. Modern Software Review: Techniques and Technologies. IRM Press, 2006. p. 368.
6. Б.У. Боэм. Инженерное проектирование программного обеспечения. М.: Радо и связь, 1985. 368 с.

REFERENCES

1. IEEE 1012-2004 *Standard for Software Verification and Validation. IEEE*, 2005. p.153. https://people.eecs.ku.edu/~hossein/Teaching/Stds/1012.pdf [in English]
2. L. Yu *A light-weight static approach to analyzing UML behavioral properties*. L. Yu, R. B. France, I. Ray,K. Lano.. Proc. of 12-th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007), pp. 56–63, 2007. p. 79. https://www.cs.colostate.edu/~iray/research/papers/iceccs07.pdf [in English]
3. M. Broy *Model Based Testing of Reactive Systems*. M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, A.Pretschner (eds.). LNCS 3472, Springer, 2005. p. 273. https://www.springer.com/gp/book/9783540262787 [in English]
4. T. *Ball Thorough Static Analysis of Device Drivers*. In Proc. of EuroSys 2006. T. Ball, E. Bounimova, B.Cook, V. Levin, J. Lichtenberg, C. McGarvey, B. Ondrusek, S. K. Rajamani, A. Ustuner., ACM SIGOPS OperatingSystems Review, 2006. p. 74. https://dl.acm.org/doi/10.1145/1217935.1217943[in English]
5. Y. K. Wong. *Modern Software Review: Techniques and Technologies*. IRM Press, 2006. p. 368 https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.174.4479&rep=rep1&type=pdf [in English]
6. B.U. Boehm. *Engineering software design*. M.: Rado and communication, 1985. 368p. http://www.library.univ.kiev.ua/ukr/elcat/new/detail.php3?doc_id=136392 [in Russian].

Надійшла у першій редакції 20.05.2020, в останній - 10.06.2020.