Model checking approach to the correctness proof of complex systems

Marina Alekseeva
P.G. Demidov Yaroslavl State University
150000 Yaroslavl, Sovetskaya 14, Russia
Email: marya_87@mail.ru

Ekaterina Dashkova
P.G. Demidov Yaroslavl State University
150000 Yaroslavl, Sovetskaya 14, Russia
Email: dea.yar@mail.ru

***Abstract*—Very often the question of efficiency in terms of execution time memory usage, or power consumption of the dedicated hardware/software systems is of utmost interest that is why different variants of algorithms are developed. In many situations the original algorithm is modified to improve its efficiency in terms like power consumption or memory consumption which were not in the focus of the original algorithm. For all this modifications it is crucial that functionality and correctness of the original algorithm is preserved [1].**

A lot of systems increasingly applying embedded software solutions to gain flexibility and cost-efficiency. One of the various approaches toward the correctness of systems is a formal verification technique which allows to verify the desirable behavior properties of a given system. This technique nowadays is well known as model checking. Model is expected to satisfy desirable properties.

Verification is the analysis of properties of all admissible program results through formal evidence for the presence of required properties. The basic idea of verifying the program is to formally prove the correspondence between the programming language and the specification of the problem.

Program and specification describe the same problem using different languages. Specification languages are purely declarative, human-centered. Imperative programming languages are more focused on executing on the computing device. Therefore less natural for men.

Likewise, this technique is an excellent debugging instrument. From the standpoint of programming technology verification enables to obtain a better strategy for debugging programs.

***Index Terms*—verification, automata-based programming, complex systems.**

I. Introduction

Correctness of Information and Communication Technology (ICT) systems [2] is the background for their safety. Errors could be catastrophic. The fatal defects in the control software are very dangerous and the number of defects grows exponentially with the number of interacting system components. Day after day ICT systems are becoming more complex.

ICT systems are universal and their reliability is the main point in the system design process. The key instrument for design process is verification techniques (fig.1). The features which are verified could be taken from specification. They are usually the main properties of the systems. They should be correct which means react adequate for any command. The accurate modelling of systems often leads to the discovery of incompleteness, ambiguities, and inconsistencies in informal system specifications.

Such problems are usually discovered at later stage of the design. The system models are accompanied by algorithms that systematically explore all states of the system model. This provides the basis for a whole range of verification techniques as model checking.

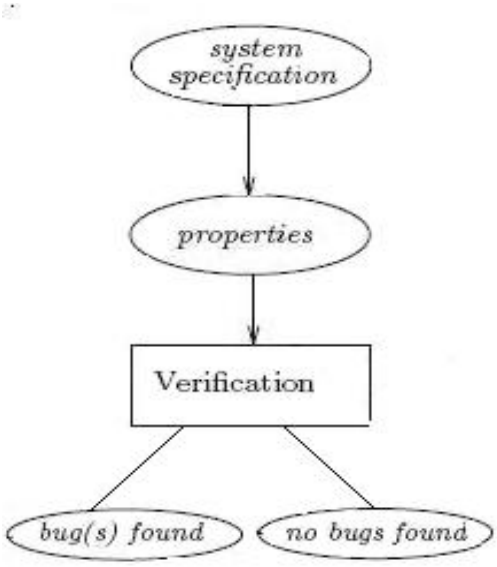

Fig. 1. The process of verification

II. Main part

A. Model-checking

Model checking [3] is one of various verification techniques. It explores all possible system states in a rude manner.

The system model is usually automatically generated from a model description that is specified in some appropriate dialect of programming or hardware description languages.

The property specification prescribes how the system behaves. All relevant system states are checked whether they satisfy the desirable property or not (fig.2).

Models of systems describe the behavior of systems in an accurate and unambiguous way. They are mostly expressed using finite-state automaton, consisting of a finite set of states and a set of transitions. In order to improve the quality of the

model, a simulation prior to the model checking can take place. Simulation can be used effectively to get rid of the simpler category of modelling errors. Eliminating these simple errors before any form of thorough checking takes place may reduce the costly and time-consuming verification effort.

Model checking has been successfully applied to several ICT systems.

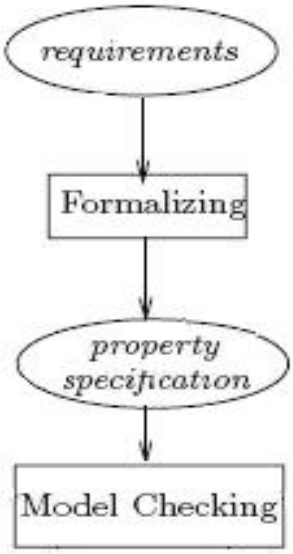

Fig. 2. The process of model-checking

B. Automata-based programming.

Automata-based programming can be used in several types of programming systems [4]:

• transforming systems (compilers, archivators). Finite automaton in programming traditionally used in design of compilers. In this situation automaton is understood as some calculating feature which has an input line and output line.

• reactive systems (telecommunication systems and systems of control and managing of physical devices). In this case the automata-based programming solves the problem of logic programming. Automaton is a device that has several parallel input lines (often binary), on which in real time the signals from the environment are coming. Processing such kind of signals, automaton is forming values for several parallel outputs.

So, the usefulness of the automata-based approach can be characterized with the combination of the words "complex behavior". For such kind of systems it is very important that automata-based approach separates the description of logic of behavior and semantics. This feature makes automaton description of complex behavior clear and understandable.

Transition systems are often used in computer science (semantical models for a broad range of high-level formalisms for concurrent systems, such as process algebras, Petri Nets, statecharts).They are a fundamental model for modelling software and hardware systems.

Transition system is defined as TS. TS is a tuple (S, Act,$\rightarrow$, I, AP, L) where

• S is a set of states,
• Act is a set of actions,
• $\rightarrow \subseteq S \times Act \times S$ is a transition relation,
• I $\subseteq$ S is a set of initial states,
• AP is a set of atomic propositions, and
• L : S $\rightarrow 2^{AP}$ is a labeling function.

TS is called finite if S, Act, and AP are finite.

Consider the following example (fig.3). The transition system in fig.3 is a schematic design of an automaton. The automaton can either deliver tea or coffee. States are represented by ovals and transitions by labeled edges. Initial states are arrow without source.

The state space is

$S = \{pay, select, tea, coffee\}$.

The set of initial states consists of only one state, i.e., $I = \{pay\}$.

The action insert coin denotes the insertion of a coin, while the automaton actions get tea and get coffee denote the delivery of tea and coffee. Transitions of which the action label is not of further interest here are all denoted by the distinguished action symbol τ. We have:

$Act = \{insert\_coin, get\_tea, get\_coffee, \tau\}$.

Automaton is represented by two locations pay (start) and select. Notes that after the insertion of a coin, the automaton nondeterministically choose to provide either coffee or tea.

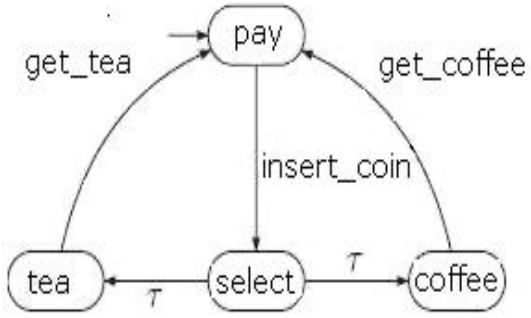

Fig. 3. A simple transition system

III. Results

Authors had an experience of applying the model checking method. Their diploma paper was devoted to the verification of the WTP (Wireless Transaction Protocol). The simple transactions were built with the help of CPN Tools and NS2 Simulator. Two types of instruments were explored.

A. System modeling. NS2.

Simulation is widely-used in system modeling for applications ranging from engineering research, business analysis, manufacturing planning, and biological science experimentation. Network Simulator (Version 2), widely known as NS2, is an event driven simulation tool which is very useful in

studying the dynamic nature of communication networks. NS2 provides users with a way of specifying such network protocols and simulating their behaviors. NS2 suggest two steps of work. The first step is constructing a model with the help of programming on C++, and finally the use of the Object-oriented Tool Command Language (OTcl) for analysis of the model and simulating the network conditions. It allows us to include our C++ programming code to the NS2 environment. We decided that NS2 is the most convenient tool for modeling the network behavior.

B. Proposed model.

The Wireless Transaction Protocol is responsible for reliable message delivery. Maximum Transfer Unit (MTU) is a maximum size of a packet in networks. If we have a message that is bigger than MTU then WTP fragmentizes this message. Flow control in cases of fragmented messages, is performed by sending fragments in groups. Every group of packets requires only one acknowledgement of the group. The last packet of each group contains a special flag. This flag indicates the end of the group and receiver knows when to send an acknowledgment. Size of each group depends on the link characteristics and the device memory. It is necessary to avoid extra packet retransmission and data loss. Receiver sends a negative acknowledgement (NAK) if the end-of-group packet is received whilst intermediate packets are missing. This operation is repeated until the entire group is received and a positive acknowledgment is sent. If timeout occurs, only the last packet of the group is retransmitted, and sender knows what packets have been lost. Wireless Transaction Protocol tries to minimize the number of unnecessary retransmissions.

In our model we have three parameters:

• t_s is the time interval between consecutive packets of the group which are sent from the sender SENDER to the receiver RECEIVER.

• t_r is the interval between consecutive packets of the group which are received by the RECEIVER.

• P_{am} as the number of packets in the group.

In our model there are two types of acknowledgments (ACK is a positive and NAK - negative acknowledgment).

When receiver sends an acknowledgment it transfers t_r with the help of it. Sender calculates special ratio. Depending on the result of this ratio sender has several situations for analysis and further actions.

• Perfect network conditions.

• Parameters can be modified by increasing P_{am}, decreasing t_s and timeout.

• There is no enough data for our algorithm to make a decision how to modify parameters (conditions of a network correspond to the established parameters).

• The network is congested, parameters can be modified by decreasing P_{am}, increasing t_s and timeout.

IV. Conclusion

Theory of programming even in the 1968 openly accepted the crisis of software development. The main symptom of this crisis is disability of the developers to provide the main feature of the software: its correctness. Theoreticians and practitioners of software underline that the crisis of methods of the development of software shows mainly during the design of the systems with complex behavior and automata-based approach can deal with this problem. That is why it is the answer for the most up-to date problems of the software development industry. The predictions show [4] that the area of applying automata-based programming will be expanded and this technology will be developed. A new models, notations and instruments will appear in the foreseeable future.

Acknowledgment

The following scientific advisers supported us by using (sometimes very) preliminary versions of this article: Valery A. Sokolov (Yaroslavl, Russia), Dmitry U. Chaly (Yaroslavl, Russia), Egor V. Kuzmin (Yaroslavl, Russia).

The authors would also like to thank the dean of Yaroslavl Demidov State University Computer Science Department P.G. Parfenov for interest and support of this project and the head of scientific-educational center "Center of Innovation Programming" Professor V.A. Sokolov for helpful advices. This work would be developed and extended in the future.

References

[1] Anikeev M., Madlener F., Schlosser A., Huss S.A., Walter C., "Automated Correctness Proof of Algorithm Variants in Elliptic Curve Cryptography" *Modeling and Analysis of Information Systems*, pp. 7–16, 2010.

[2] Baier Christel, Katoen Joost-Pieter. "Principles of Model Checking," *The MIT Press, Cambridge, Massachusetts, London, England*, 2008.

[3] Egor V. Kuzmin, "Introduction to the theory of mathematical processes and structures," *Yaroslavl Demidov State University, Yaroslavl, Russia*, 2001.

[4] N.I. Polikarpova, A.A. Shalyto, "Automata-based programming" *Saint-Petersburg State University of Informatic Technologies, Mechanics and Optic, Saint-Petersburg, Russia*, 2009.