Model Checking Approach to the Correctness Proof of Complex Systems

Marina M. Alekseeva, Ekaterina A. Dashkova
Yaroslavl Demidov State University
150000 Yaroslavl, Sovetskaya 14, Russia
{marya_87, 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 for desired behavioral properties of a given system to be verified. 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 the reliability of ICT systems is the main point in the system design process. The key instrument for design process are 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 modeling 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.

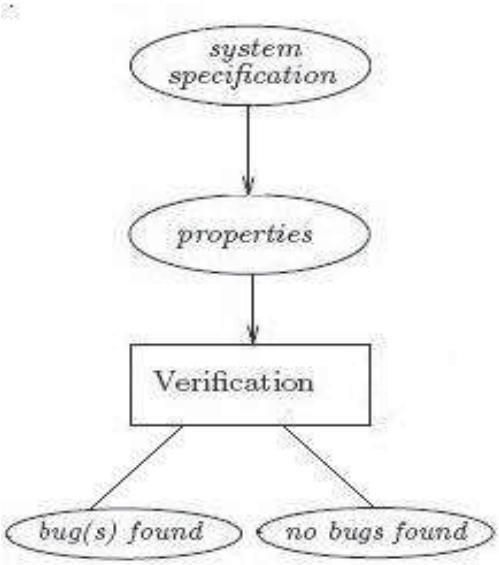

Fig. 1. The process of verification

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 desired 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 automata, 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 modeling errors. Eliminating these simpler 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.

B. Automata-based programming

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

• transforming systems (compilers, archivators). Finite automata in programming traditionally used in design of compilers. In this situation automata 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. In this case automaton is a device that has several parallel input lines (often binary), on which in real time the signals from the environment is coming. Processing such kind of signals, automaton is forming values for several parallel outputs.

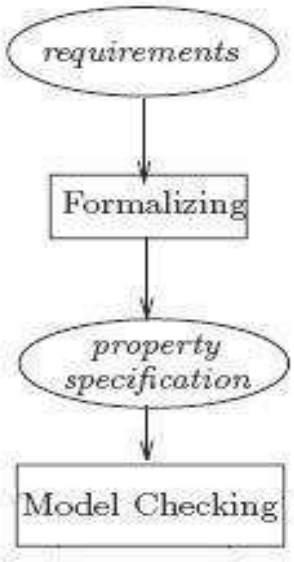

Fig. 2. The process of model-checking

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 automata 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 modeling 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 start and select. Notes that after the insertion of a coin, the automaton nondeterministically choose to provide either coffee or tea.

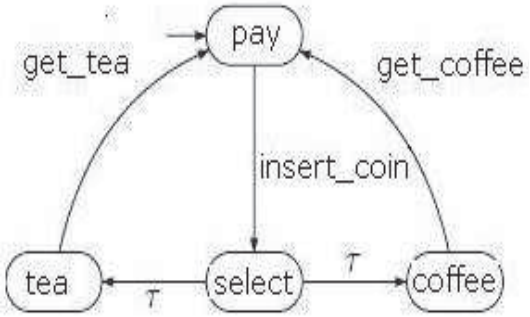

Fig. 3. A simple transition system

III. Conclusion

Theory of programming even in the 1968 openly accepted the cricis 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 expand and this technology will be develop. 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 Criptography” *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.