A Method for Automatic Runtime Verification of Automata-Based Programs

Oleg Stepanov, Anatoly Shalyto (research supervisor)
Fac. of Information Technologies and Programming
St. Petersburg State University of Information Technologies, Mechanics and Optics
Email: oleg.stepanov@gmail.com, shalyto@mail.ifmo.ru

***Abstract*—Currently Model Checking is the only practically used method for verification of automata-based programs. However, current implementations of this method only allow verification of simple automata systems. We suggest using a different approach, runtime verification, for verification of automata systems. We discuss advantages and disadvantages of this approach, propose a method for automatic verification of automata-based programs which uses this approach and conduct experimental performance study of the method.**

***Index Terms*—automata, software verification, runtime verification, alternating automata**

I. Introduction

Program verification — checking that a program satisfies specified constraints [1] — is considered an important problem. Two approaches for verification are currently used: static verification and runtime verification. A static verification method, namely *Model Checking* [2], is the widely used approach to verification. Model Checking is a verification approach based on analysis of models of a program. For this method to be used the program must be presented in a special form, a *Kripke structure* [2], which describes possible changes of program's computational state. This structure is related to the main method's drawback: size of the structure grows exponentially with linear increase of the program size. This problem is known as "exponential blowup".

Another approach for verification, *runtime verification*, is used to verify a program's behavior at runtime, e.g. when there's no access to the program source code or to verify interactions between programs. Runtime verification is an approach for verification which takes traces of program runs and checks them against the specification. Although verification of a particular program run cannot guarantee satisfaction of the specified constraints, it can give reliable results if one chooses input data carefully. Another advantage of runtime verification is that its complexity does not depend on complexity of the verified program; it only depends on complexity of the specification and size of the program trace. Finally, runtime verification is performed on the program itself and not on its model thus avoiding possible mismatches between the program and the model which can occur when using Model Checking.

Static runtime verification uses various temporal logics the specification language. The most commonly used of the are *linear temporal logic* (LTL) and *computation tree log* (CTL).

Verification of automata-based programs to date us Model Checking as its primary tool. It is a promising directi because models of automata-based programs can be bu automatically with little or no errors at all and these mode can be formally proven to match the source program. Als computational state of automata-based systems is typical relatively small so researchers' best hopes were that it w small enough to perform static verification in considerab time of modern commodity hardware.

An important research is being done now ([2], [3]), a part which is development and analysis of various approaches static verification of automata-based programs. They list number of different methods using a great variety commercially available and custom developed verificati tools some of which are operating directly on the automa model. However, current implementations are very limited capabilities, and only allow verification of models wi hundreds of transitions in reasonable time.

Complete methods for runtime verification of automat based programs are virtually unknown. At the same tim runtime verification of imperative programs is studied w and has received significant attention recently. Our intent is pick an available approach to runtime verification and apply to verification of automata-based programs. Since automat based programs are usually formally defined, it is usual expected for a new approach to be introduced as a form method.

The paper presents a method for runtime verification automata-based programs based on a known approach runtime verification [4]. The presented method is based traversal of *alternating automata* [5]. This allows verificati of program traces with computational complexity that linear depends of trace size and complexity of the specification. T primary components of the proposed methods are the set propositions allowed in the specifications, the algorithm f building the traces and the rules to evaluate values of t propositions at each point of the trace.

An important constraint of runtime verification is thet results are unreliable when the method is applied to progra containing blocks that execute in parallel. This is so becau the order of execution of parallel blocks is nondeterminis and can change from one run to another. In this paper w