Science of Computer Programming 103 (2015) 51–70

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Efficient safety checking for automotive operating systems using property-based slicing and constraint-based environment generation ☆

Yunja Choi *, Mingyu Park, Taejoon Byun, Dongwoo Kim

School of Computer Science and Engineering, Kyungpook National University, Daegu, South Korea

ARTICLE INFO

Article history:
Received 1 June 2013
Received in revised form 16 October 2014
Accepted 20 October 2014
Available online 24 October 2014

Keywords:
Safety
Slicing
Model checking
Testing
Automotive OS

ABSTRACT

An automotive operating system is a safety-critical system that has a critical impact on the safety of road vehicles. Safety verification is a must in each stage of software development in such a system, but most existing work focuses on specification-level or model-level safety verification. This work proposes a collaborative approach using model checking and testing for the efficient safety checking of an automotive operating system. Efficiency is achieved through property-based slicing, which reduces the complexity of verification, and guided test sequence generation, which limits the input space to a set of representative test sequences selected from legal as well as illegal input spaces. Comprehensiveness is achieved by formally specifying external constraints using constraint automata from which guided test sequences are selected. The approach is implemented as a prototype tool set applied to the verification of an open source automotive operating system based on the OSEK/VDX international standard. The approach revealed several safety issues that could not be identified by existing approaches.

1. Introduction

As most safety-critical systems are increasingly controlled by software, software safety is becoming a prerequisite for system safety that must be thoroughly checked. A representative example can be found in the automotive industry, where road vehicles are equipped with an increasing number of electrical devices called ECUs (Electrical Control Units), which are controlled by the operating system. The operating system is the core of the control software and any malfunction on its part can cause critical errors in the automotive system, which in turn may result in loss of lives and assets.

The difficulty in software safety checking lies in its high complexity. No matter whether it is for functional safety or code safety, safety checking for software requires comprehensive behavioral analysis, which often ranges over hundreds of millions of different cases. The complexity remains the same regardless of the choice of verification technique: either manual analysis, automated verification using dynamic testing, or static model checking. Testing has been widely used as a systematic and cost-effective safety analysis/assurance method [2,3], but its optimistic incompleteness often misses critical problems and cannot guarantee the “absence of wrong behavior” unless testing can be exhaustively performed on every possible execution path of the software, which is costly, if not impossible. Model checking [4,5] is an alternative comple-

☆ An earlier version of this paper has been published in FTSCS 2012 [1].

* Corresponding author.
E-mail addresses: yuchoi76@knu.ac.kr (Y. Choi), pqrk8805@gmail.com (M. Park), bntejn@gmail.com (T. Byun), kdw9242@gmail.com (D. Kim).

http://dx.doi.org/10.1016/j.scico.2014.10.006
0167-6423/

mentary verification technique that, in a sense, automatically performs exhaustive testing. It is suitable for comprehensive functional safety analysis and can effectively identify subtle issues, such as process deadlock, illegal functional behavior, and starvation. However, its exhaustiveness naturally requires more resources and can often be too costly to be practical. Reducing verification cost in practice requires expertise in formal methods as well as domain knowledge.

A practical solution to this problem has been actively sought with various abstraction and engineering techniques [6–9]. Nevertheless, efficient approaches for safety checking embedded software are relatively rare, especially for automotive operating systems. This may be partly due to the fact that the operational environment plays a crucial role in the verification of operating systems, but it is not trivial to comprehend the environment.

Based upon our experiences with safety analysis for automotive operating systems [10], we expect that successful safety checking for automotive software requires solutions to the following issues:

1. The size of the model/code to be verified needs to be minimized to avoid state-space explosion.
2. Efficient modeling of the environment, such as user tasks and hardware environment, is necessary and critical.

Since an operating system is a reactive system responding to environmental stimuli, the correctness of its behavior needs to be analyzed with respect to the behavior of its environments. A selected representative environment is often used in testing in the form of test drivers, but its comprehensiveness is hard to justify. A non-deterministic environment is typically used to over-approximate actual behavior in model checking, but it is often too expensive for exhaustive verification. The difficulty and importance of defining a *good* environment model has been addressed in a number of previous works [11–16].

We note that these two problems apply to both model checking and testing. Though the level of comprehensiveness differs, both techniques rely on automated search techniques initiated by environmental stimuli. This is called environment model in model checking and test scenario in testing. This work proposes a solution to these two problems using property-based code slicing, to reduce the size of the code to be verified and to generate an efficient and effective environment model. The goal is to minimize the kernel code to the set of functions relevant for a given property and to construct a comprehensive usage model for functional safety checking. The sliced code together with its environment model is verified using both testing and model checking to compare their impact on verification efficiency. Our code slicing is an extension of existing techniques [17,18] with more emphasis on function slicing where inter-procedural call relations and the use of global variables need to be considered.

Function slicing extracts a set of compilable functions that have direct/indirect dependencies on a given property to be verified, by (1) identifying variables involved in the property (target variables, initial slicing criteria), (2) identifying all the statements and relevant variables that directly/indirectly modify the target variables (extended slicing criteria), (3) identifying functions that modify global variables that are directly/indirectly relevant for the target variables, and (4) constructing a minimal compilable code fragment including all the statements directly/indirectly relevant for the extended slicing criteria.

A list of system-level functions identified during the slicing process is used to construct a valid and comprehensive environment model for an operating system in the form of constraint automata. A constraint automaton is a formal specification of an external constraint specified in the OSEK/VDX [19] standard for automotive operating systems. The formal specification allows us to perform a guided search through valid/invalid input spaces and to construct an environment that subsumes representative input sequences of the entire input space. The environment model is then used to comprehensively generate system-level test sequences. The same set of constraint automata is used to generate module-level test sequences through the mapping between system-level API functions and module-level function sequences. Comprehensive verification of the extracted set of functions can be performed using both testing and model checking.

Software slicing [18] is widely known and used in various program analysis techniques [20,21,17,22,23]. The novelty of our approach lies in the use of the slicing result for environment modeling. Formal constraint specification and generation of test sequences for automotive operating systems are also new. Several improvements have been made compared to the earlier version of this paper presented at FTSCS 2012 [1], including the following major changes:

1. A rigorous process for inter-procedural property-based slicing is defined and implemented.
2. Environment modeling is formulated using constraint automata.
3. Verification efficiency has been improved by guided test sequence generation.
4. Module-level internal constraints are systematically identified.

The approach and the tool are applied to the verification of assertions of the Trampoline operating system [24], which is an open source automotive operating system compliant with OSEK/VDX. The model checker CBMC [25] and system-level/module-level guided testing were used to compare their fault-detection capability, their comprehensiveness in terms of code coverage, and their efficiency in terms of resource consumption.

The remainder of this paper is organized as follows. Section 2 provides the motivation for our work, followed by an overview of our approach in Section 3. Section 4 introduces our property-based function slicing technique and Section 5 presents methods for constructing environment models and for automated test generation. Section 6 explains the implementation details. Section 7 presents the experimental results and the evaluation using the Trampoline OS as a case example. We conclude in Section 9, after a discussion on related work in Section 8.