Science of Computer Programming 163 (2018) 19–41

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

A two-step approach for pattern-based API-call constraint checking ☆

Dongwoo Kim, Yunja Choi *

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

ARTICLE INFO

Article history:
Received 31 July 2017
Received in revised form 31 March 2018
Accepted 4 April 2018
Available online 9 April 2018

Keywords:
API
Constraint checking
Automotive software
Model checking
OSEK/VDX

ABSTRACT

An operating system publishes a set of application programming interface (API) functions along with a set of API-call constraints with which programs running on the operating system must comply. Any violation of these constraints may become a source of massive property damage or even human injury when such a program is used for safety-critical systems. A rigorous and targeted verification method is needed to identify such violations, which are frequently subtle and difficult to identify using conventional verification methods.

As automated tool support for pre-checking constraint violations in the development process, this study presents a two-step approach for checking API-call constraints by using predefined patterns specifically designed for automotive operating systems. A lightweight checking method is designed for quick-and-easy checking of local API-call constraints, which utilizes constraint patterns and the C code model checker CBMC. The global constraint checking method is a heavyweight method, as it requires behavior models of the underlying operating system constructs as well as constraint patterns, but it produces more accurate verification results; it uses the symbolic model checker NuSMV as the backend verification engine. This two-step approach is effective in identifying constraint violations and efficient in reducing false alarms from infeasible execution paths.

© 2018 Elsevier B.V. All rights reserved.

1. Introduction

In general, embedded control software consists of two tightly coupled components: a control program and an operating system. The verification of such control software requires checking its individual components as well as the interactions between them, particularly when the software is designed for controlling a safety-critical system. In the automotive domain, for example, a modern car is equipped with electrical devices called electronic control units (ECUs), each of which is dedicated to controlling a critical hardware component for operating an automobile, such as the brake control module, the engine control module, and powertrain control module. Moreover, each of these ECUs is controlled by different software programs, which are composed of a common operating system and a control program. Typically, a common operating system is used for all ECUs developed according to international standards, such as OSEK/VDX [3] or AUTOSAR [4]; however, a different control program is developed for each ECU depending on its role.

☆ This paper is an extended version of [1] and [2].

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

https://doi.org/10.1016/j.scico.2018.04.001
0167-6423/© 2018 Elsevier B.V. All rights reserved.

To decouple these two components while providing support for seamless communication between them, the operating system (OS) publishes a set of API functions along with a set of API-call constraints that should be followed by control programs running on the OS. A violation of these constraints may become a source of system failure, possibly leading to massive property damage or human injury [5]. However, such issues are frequently subtle and difficult to identify using conventional verification methods, which focus more on functional verification than on the safety of the interactions.

As automated tool support for pre-checking constraint violations in the development process, a two-step approach is presented in this study for checking API-call constraints in the software. The goal is to identify constraint violations at the application program level before it is compiled with its underlying operating system. As a quick-and-easy verification method, the first step targets only *local constraints*, which can be checked independent of the behavior of the OS. Similar to unit testing, it checks each task of a given application program individually. The second step uses a heavyweight but more rigorous verification method aimed at checking *global constraints*, which do involve the behavior of the OS.

The first step is achieved by modeling constraints as C library functions for use in monitoring API function calls and by using the C code model checker CBMC [6] as its underlying checking engine to verify the compliance of API function calls with respect to the given constraints. The second step lifts the level of abstraction for the verification by utilizing formal OS patterns and constraint patterns defined in our previous works [7,8] and by converting an application source code written in C into a synchronized composition of state machines, which are all specified in the input language of NuSMV [9]. This verification model consists of an OS model, which is auto-generated from a given system configuration, and a set of formal OS patterns; an application model converted from application programs written in C; and a constraint model, which is also auto-generated from the same configuration information and a selected set of constraint patterns. This verification model is then verified using the model checker NuSMV to determine whether the constraint model remains in a safe state.

Our approach is unique in the sense that the formal models are auto-constructed from two sets of predefined patterns: one for the OS constructs and the other for the API-call constraints. The patterns for the API-call constraints were first introduced in [10] and have been elaborated and utilized in automated test generation [5,8]. Our two-step approach also utilizes constraint patterns to generate the monitoring code (for local constraint checking) and to specify the verification properties for the linear time logic (LTL) model checking (for global constraint checking). The patterns for the OS constructs and the general framework for the configurable model generation for the OSEK/VDX OSs are defined in [7]. We used the prototype tool introduced in that paper to auto-generate OS models for checking global constraints.

We conducted a series of experiments using four sets of applications generated from our test generator [5], a conformance test suite for automotive software, a game application [11], and a set of benchmark applications for the Erika [12] OS to evaluate the detection capability and performance of the two-step verification approach. These experiments showed that the suggested approach can identify most of the violations of the API-call constraints with a low rate of false alarms. Through manual analysis and visualized simulation, we identified 30% of the reported counterexamples from the global constraint checking as false alarms, whereas the local constraint checking did not produce any false alarms. While the cost for the lightweight constraint checking using CBMC was negligible, the performance of the heavyweight global constraint checking was largely affected by the number of tasks in the application program.

This work is an extension of [2], which introduced local constraint checking, and of [1], which introduced global constraint checking. Major additions in this work include (1) the two-step approach that integrates the two existing techniques into one verification framework; (2) the formalization of the state machine representation of the application code, specifically designed for global constraint checking; and (3) various experiments on real examples to evaluate the suggested approach.

The remainder of this paper is organized as follows. Sections 2 and 3 present the background of this study and an overview of the suggested approach, respectively. After a brief summary of API-call constraint patterns in Section 4, each step of the two-step approach is explained in Sections 5 and 6. A prototype implementation of the suggested method and a series of experiments are presented in Section 7. Following a brief discussion of related work in Section 8, we conclude with a discussion of our findings and future work in Section 9.

2. Background

2.1. OSEK/VDX

A modern car consists of a number of components, such as an engine module, a brake module, headlights, etc., each of which is controlled by an ECU. An ECU is a small computing device that has its own CPU and memory space and is controlled by embedded software. Typically, the control software on an ECU is produced by compiling an OS, the system configuration, and the application logic. As OSs and application software are developed independently by different organizations, interfaces are internationally standardized to avoid problems in the integration process.

The OSEK/VDX OS is a de-facto standard for automotive control software. It aims at an industry standard for an open-ended architecture for distributed control units in vehicles [3]. In particular, the OS part of its specification has been adopted by the AUTOSAR [4] consortium and is being widely used in the automotive industry worldwide.

An OS that is compliant with OSEK/VDX is typically written in C or an assembly language, and implement a set of system services along with a set of API functions that can be used by application programs. According to the OSEK/VDX specifications, an OS consists of a number of basic constructs, including tasks, events, resources, alarms, and interrupt

Download English Version:

https://daneshyari.com/en/article/6875173

Download Persian Version:

https://daneshyari.com/article/6875173

Daneshyari.com