Science of Computer Programming 127 (2016) 76–102

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Algorithmic verification of procedural programs in the presence of code variability

Siavash Soleimanifard *, Dilian Gurov *

KTH Royal Institute of Technology, Stockholm, Sweden

ARTICLE INFO

Article history:
Received 30 January 2015
Received in revised form 24 August 2015
Accepted 26 August 2015
Available online 16 September 2015

Keywords:
Compositional verification
Model checking
Maximal models

ABSTRACT

We present a generic framework for verifying temporal safety properties of procedural programs that are dynamically or statically configured by replacing, adapting, or adding new components. To deal with such a variability of a program, we require programmers to provide local specifications for its variable components, and verify the global properties by replacing these specifications with maximal models. Our framework is a generalization of a previously developed framework that fully abstracts from program data. In this work, we recapture program data and thus significantly increase the range of properties that can be verified. Our framework is generic by being parametric on the set of observed program events and their semantics. We separate program structure from the behaviour it induces to facilitate independent component specification and verification. To exemplify the use of the framework, we develop three concrete instantiations; in particular, we derive a compositional verification technique for programs written in a procedural language with pointers as the only datatype.

1. Introduction

In modern computing systems code changes frequently. Components evolve rapidly or exist in multiple versions customized for different users, and in open and mobile contexts a system may even automatically reconfigure itself. As a result, systems are no longer developed as monolithic applications; instead they are composed of ready-made off-the-shelf components, and each component may be dynamically replaced by a new one that provides improved or additional functionality. The design and implementation of systems with such static and dynamic *variability* has been attracting considerable attention over the past years. However, there has been less attention on their formal verification. In this paper, we develop a generic framework for the verification of temporal safety properties of such systems.

The verification of *variable systems* is challenging because the code of the variable components is either not available at verification time or changes frequently. Therefore, an ideal verification technique for such systems should (i) *localize* the verification of variable components, and (ii) *relativize* the *global* properties of the system on the correctness of its variable components. This can be achieved through a compositional verification scheme where system components are specified *locally* and verified independently, while the correctness of its global properties is inferred from these local specifications, thus allowing an independent evolution of the implementations of individual components, only requiring the re-establishment of their local correctness.

* Corresponding authors.
E-mail addresses: siavashs@csc.kth.se (S. Soleimanifard), dilian@csc.kth.se (D. Gurov).

http://dx.doi.org/10.1016/j.scico.2015.08.010
0167-6423/

One algorithmic technique that realizes the above verification scheme is to replace the local specifications of the variable components by so-called *maximal models* [19], and then to model check the resulting system against the global properties of interest. Maximal models are the most general program models (for the purposes of verifying global properties from a given property class) that satisfy the corresponding local specifications. Such models are only guaranteed to exist and be unique in well-chosen set-ups (like the present one). The two main bottlenecks of the technique are (i) the high cost of computing maximal models from local specifications, and (ii) the added burden to the programmer of producing component specifications.

In previous work (see e.g. [21,23,20]) we developed a maximal model based technique for the verification of temporal safety properties of procedural programs, based on a program model that focuses purely on control flow and abstracts away all program data. This rather drastic abstraction was chosen as a first step in the realization of the compositional verification scheme discussed above, and was motivated by considerations of efficiency of maximal model construction and model checking (notice that the state space can still be infinite due to possible unbounded recursion), and of easiness of specification. It made possible the fully automated tool-based verification of procedural programs for control flow based temporal safety properties (see e.g. [35]). An example of such a property over sequences of method invocations that does not involve data is: "method m_1 is not called after method m_2 is called".

Abstracting away from all program data, however, significantly impacts the range of properties that can be handled. Properties that involve program data, such as "method m_1 is called only if variable `V` is not pointing to `null`", cannot be specified and verified. In the present work, we generalize our previous technique to re-capture program data, thus bringing the usability of our work to a whole new level. This generalization has to be performed in a controlled fashion, carefully avoiding unnecessary computational blow-up, as we show in Section 2. Incorporating program data, if done in the straightforward fashion, makes the maximal model construction and property specification impractical: the program models and properties become too detailed and large, maximal model construction becomes unmanageably complex, and the program models become overly specific to the given programming language. Our present proposal re-captures program data without adding extra complexity to the maximal model construction, and keeps the effort of property specification within practical limits.

We propose a control flow model that is parametric on a set of *actions* that model (observable) user-selected instruction types, and on Hoare-style state *assertions* that capture abstractly the effect of sequences of (unobservable) statements between consecutive actions. We combine the abstraction provided by assertions with the precision provided by actions to define a uniform control flow graph representation of programs that can be tuned to the verification of the properties of interest.[1] The abstraction provided by assertions avoids the local specifications to become overly verbose, and allows to capture program data without adding complexity to the maximal model construction.

We present three instantiations of our generic verification framework. The first one abstracts away all data, showing the presented framework to be a proper generalization of the original framework [21]. The second instantiation is for *Boolean programs* [10], illustrating how our generic framework can handle data from finite domains. In the third and most challenging instantiation we exemplify the use of our framework for the verification of programs written in a procedural language called PoP [33] that has pointers as the only datatype. Dealing with this language is challenging because, in addition to unbounded call stacks, it can give rise to infinite state spaces for yet another reason, namely unbounded pointer creation. Still, the model checking problem has been shown to be decidable for PoP programs. This instantiation shows how our framework can cope with data from certain infinite domains.

The main *contributions* of the present paper are: (i) a novel control flow model that combines the precise ordering of selected types of instructions with an abstract representation of the remaining irrelevant ones, together with an operational semantics (a behavioural model), (ii) an adaptation of the original maximal model construction to the case with data (possibly from infinite domains) with minimal additional cost, (iii) a proof of the correctness of the technique by a (non-trivial) re-establishment of our previous results, and (iv) three instantiations of the generic framework, most notably for a procedural language with the pointer datatype (PoP). A short version of the present paper appeared in [34].

Organization of the paper Section 2 provides an overview of our technique and illustrates some variability scenarios on an example. Sections 3, 4, and 5 define our program models, specification languages, and maximal models. In Section 6 we spell out our compositional verification principle. Throughout these sections we shall use the instantiation of the generic framework to the case with full data abstraction to illustrate the main notions. Sections 7 and 8 present two other instantiations of our framework, namely for Boolean programs and for Pointer programs, respectively. Finally, in the last two sections we discuss related work and draw conclusions.

2. Overview of the approach

This section provides an overview of our framework by demonstrating its use on an example that mimics the method invocation style of real-life web applications. Although the technique we propose applies to procedural languages in general,

[1] From a wider perspective not further explored here, providing Hoare-style assertions and precise ordering of actions opens the ground for combining Hoare-style verification with temporal logic reasoning.