Model checking software product lines with SNIP

Andreas Classen · Maxime Cordy · Patrick Heymans · Axel Legay · Pierre-Yves Schobbens

Published online: 14 June 2012
© Springer-Verlag 2012

Abstract We present SNIP, an efficient model checker for software product lines (SPLs). Variability in software product lines is generally expressed in terms of features, and the number of potential products is exponential in the number of features. Whereas classical model checkers are only capable of checking properties against each individual product in the product line, SNIP exploits specifically designed algorithms to check all products in a single step. This is done by using a concise mathematical structure for product line behaviour, that exploits similarities and represents the behaviour of all products in a compact manner. Specification of an SPL in SNIP relies on the combination of two specification languages: TVL to describe the variability in the product line, and fPromela to describe the behaviour of the individual products. SNIP is thus one of the first tools equipped with specification languages to formally express both the variability and the behaviours of the products of the product line. The paper assesses SNIP and suggests that this is the first model checker for SPLs that can be used outside the academic arena.

Keywords Model checking · Product lines · Tool · Language · Feature

1 Introduction

Software product line (SPL) engineering is an increasingly popular software development paradigm for building families of similar software products. Many of those software products are used in critical applications such as automotive or avionics. This requires a solid evidence that they indeed work correctly with respect to their requirements and intended properties. For example, a desired property for the file transfer protocol studied in Sect. 6 is: “A sent message eventually reaches its destination”.

A simple but cumbersome approach for product line verification consists in applying classical model checking algorithms [38] on each individual product of the family. However, for an SPL with $n$ features, this would lead to $2^n$ calls of the model checking algorithm. This solution is clearly unsatisfactory and should be replaced by new approaches that take the variability within the family into account. Those approaches often rely on compact mathematical representations on which a specialized model checking algorithm can be applied. The main difficulties are (1) to develop such a model checking algorithm, and (2) to propose mathematical structures that are compact and flexible enough to take the variability of the family and its specification into account.

In [13], we introduced featured transition systems (FTSs), an extension of transition systems used to represent the behaviour of all the products of an SPL in a single compact structure. We also proposed new model checking algorithms that make use of the compact structure of FTSs to verify the whole SPL in a single execution. More precisely, these
semi-symbolic algorithms model-check the SPL against temporal properties expressed in linear temporal logic (LTL) [35]. Those algorithms, capable of identifying all the products of the SPL that do satisfy a property, are called FTS algorithms.

This article presents SNIP, an SPL model checking tool that implements our theory. Concretely, SNIP implements the FTS algorithms for verifying SPLs against LTL properties. Not only does SNIP put these theoretical results into practice, it also reflects our concern of combining them with high-level specification languages for product line specification—a mandatory step to transfer our results from theory to practical applications. More precisely, the specification of the SPL in SNIP relies on the combination of two specification languages: TVL [11] to describe the variability in the family, and fPromela to describe the behaviour of the individual products in a compact manner.

fPromela is an extension of Promela, the modelling language used by the well-known model checker SPIN [23]. The syntax of Promela is close to procedural programming languages such as Pascal or C. Its relatively low complexity makes it an easy-to-learn modelling language. fPromela extends Promela by adding a guard operator, which allows to make a transition available only to a subset of the products. TVL is a text-based feature modelling language that allows to declare the features of an SPL, as well as the mathematical relations and constraints between them. Hence, the TVL model defines the set of products of the SPL that are valid with respect to the specified constraints.

Given an LTL property and an SPL described in TVL and fPromela, SNIP uses the FTS algorithms to model check the whole product line instead of each product individually. In case there are products that do not satisfy the property, the tool will identify one (or all) of these product(s) together with a counterexample. To the best of our knowledge, SNIP is the only SPL model checking tool with a high-level specification language that is able to model check SPLs in an efficient manner.

In addition to the FTS algorithms, SNIP also implements a naive approach in which each product is model checked individually. This permits us to compare the two approaches and assess the efficiency of the FTS algorithms. This is done by measuring the time needed by both algorithms implemented in SNIP to verify the SPL of a file transfer protocol against several properties.

Structure of the paper. Section 2 surveys related work. Section 3 presents the theoretical foundations of SNIP. Section 4 gives the syntax and semantic of fPromela and its relation to FTSs, hence showing soundness and correctness of the implementation. Section 5 describes the user interface of SNIP and discusses its architecture. Finally, Sect. 6 presents the setup and the results of a case study.

2 Brief overview of related work

SNIP is one of the first tools for SPL model checking, which means that there are only few approaches that can be compared to what we propose in this paper. For a thorough discussion of the wider (mostly theoretical) related work, the interested reader is referred to [14].

Let us begin with a brief overview of SPL model checking before we discuss tools related to SNIP. A number of fundamental models for product line behaviour have been proposed in the literature. Most of those proposals do not embed any verification procedure. Among such existing approaches, one finds those based on UML, e.g. [16,28,33,39], which extend existing UML diagram types (sequence diagrams, state machines) to allow them to model SPL behaviour. However, verification can only be performed after a model of a specific product has been derived. This leads to a scalability problem as the number of products is potentially huge.

More formal approaches to modelling SPL behaviour are based on modal transition systems (MTSs) [18,30] and modal I/O automata [31]. In these approaches, transitions can be mandatory (required transitions) or optional (allowed transitions). As expected, allowed transitions can be used to model variability, and an MTS essentially specifies a family of behaviours. In addition to basic MTSs, there are proposals to extend them by introducing variability operators to specify cases in which a specific number of outgoing transitions may be taken [17]. In [2,3], the authors propose a deontic logic interpreted over MTSs, that can be used to express both behavioural properties, and constraints over features. The CCS process algebra was extended in a similar way [21], with an operator that allows to model variability in the form of alternative choice between two processes.

All these approaches are severely limited in that they cannot relate a behaviour of a model to a specific product or feature of the SPL. For example, MTSs are not capable to capture the features that make a transition optional; similarly, choices between processes in [21] are not linked to features. In consequence, a verification algorithm cannot identify bad products (those that violate a property) or interacting features. Our recent work [12–14] as well as [32] overcome this limitation by explicitly linking transitions to features. The key idea of linking transitions to features is transposed as-is to SNIP, the tool we focus on in this paper.

There are a number of tools similar to SNIP. In our earlier work [12], we developed an extension of the NuSMV model checker [8] which can model check SPLs expressed using the fSMV language [34]. This tool uses the fully symbolic FTS algorithm from [12], whereas SNIP uses the semi-symbolic algorithms from [13,14]. SNIP differs in other ways from our NuSMV extension. Its modelling language, fPromela, is based on annotation rather than composition [26].