Science of Computer Programming 80 (2014) 416–439

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Formal semantics, modular specification, and symbolic verification of product-line behaviour ☆

Andreas Classen [a], Maxime Cordy [a,*,1], Patrick Heymans [a,b], Axel Legay [c], Pierre-Yves Schobbens [a]

[a] PReCISE Research Center, University of Namur, Belgium
[b] INRIA Lille-Nord Europe, Université Lille 1 – LIFL – CNRS, France
[c] INRIA Rennes, France

HIGHLIGHTS

- We use Featured Transition Systems (FTS) to model Software Product Lines (SPLs).
- We design *symbolic* algorithms for checking an FTS against temporal properties.
- We give a new compositional formal semantics to the fSMV language.
- We prove the expressiveness equivalence between fSMV and FTS.
- We evaluate practical implications of our results through our toolset and case study.

ARTICLE INFO

Article history:
Received 5 May 2012
Received in revised form 22 August 2013
Accepted 18 September 2013
Available online 22 October 2013

Keywords:
Software product line
Verification
Feature
Language
Specification

ABSTRACT

Formal techniques for specifying and verifying Software Product Lines (SPL) are actively studied. While the foundations of this domain recently made significant progress with the introduction of Featured Transition Systems (FTSs) and associated algorithms, SPL model checking still faces the well-known state explosion problem. Moreover, there is a need for high-level specification languages usable in industry. We address the state explosion problem by applying the principles of symbolic model checking to FTS-based verification of SPLs. In order to specify properties on specific products only, we extend the temporal logic CTL with feature quantifiers. Next, we show how SPL behaviour can be specified with fSMV, a variant of SMV, the specification language of the industry-strength model checker NuSMV. fSMV is a feature-oriented extension of SMV originally introduced by Plath and Ryan. We prove that fSMV and FTSs are expressively equivalent. Finally, we connect these results to a NuSMV extension we developed for verifying SPLs against CTL properties.

1. Introduction

Software Product Lines (SPLs) are a popular software engineering paradigm that seeks to maximise reuse by planning upfront which *features* should be common, resp. variable, for several similar software systems [17]. The different systems in an SPL (called "products") are identified in advance and a model of their differences and commonalities is created. This model is usually a *feature diagram* [29,39], features being atomic units of difference that appear natural to stakeholders and

☆ This article is an extended version of the paper *A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, Symbolic model checking of software product lines, in: Proceedings of ICSE '11, ACM, 2011, pp. 321–330.*

* Corresponding author.
E-mail addresses: acs@info.fundp.ac.be (A. Classen), mcr@info.fundp.ac.be (M. Cordy), phe@info.fundp.ac.be (P. Heymans), axel.legay@inria.fr (A. Legay), pys@info.fundp.ac.be (P.-Y. Schobbens).

[1] FNRS research fellow, project FC 91490.

0167-6423/$ – see front matter
http://dx.doi.org/10.1016/j.scico.2013.09.019

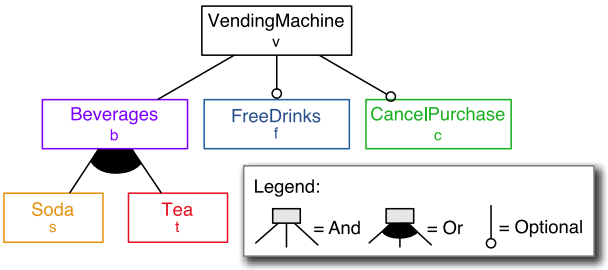

Fig. 1. The feature diagram of a vending machine.

technicians alike [13]. A toy example of a feature diagram is given in Fig. 1. It models a vending machine SPL with five features. Feature *Beverages* is mandatory, while *FreeDrinks* and *CancelPurchase* are optional. *Soda* and *Tea* are subfeatures of *Beverages*, and any product must have at least one of them.

In the real world, SPL development is increasingly applied to embedded and critical systems [21]. Formal modelling and verification of SPL behaviour are thus vital for quality assurance and are actively studied [24,31,32,15]. Model checking is a well-known automatic technique for verifying both hardware and software. It allows to verify desired behavioural properties on a model of a given system [4]. For example, an intended property for the vending machine is: "A customer can always cancel a purchase before the beverage is served". In the context of single systems, a model checker returns true if a property is satisfied, or a counter-example (i.e. an execution trace) if it is violated. The model checking problem in SPL is different from the one in single systems engineering: an algorithm has to check all products against a property and pinpoint those products that violate it [16]. In our example, every vending machine without the *Cancel* feature would not satisfy the aforementioned property. The model checking problem is also harder, as it has to deal with the fact that there can be exponentially many, $O(2^{\#features})$, products to verify. In [15,9], we addressed the model checking problem for SPLs by introducing *Featured Transition Systems* (*FTSs*), a formalism to express the behaviour of all products of the SPL in one model. FTSs are transition systems [4] in which transitions are labelled with features (in addition to being labelled with actions). This allows one to keep track of the different products. We also proposed new model checking algorithms [15] that exploit the structure of the FTS and try to avoid an exponential number of verifications by exploring the FTS rather than the transition system of each individual product. Those algorithms can be used to verify properties expressed in Linear Time Logic (LTL). We call them *FTS algorithms*. The experimental results gathered so far show that this new approach is more efficient than an enumerative method that verifies each product individually. More information can be found on our project website http://www.info.fundp.ac.be/fts.

The main drawback of our previous FTS algorithms is that they rely on an explicit enumeration of the state space. Albeit we already observed that FTSs drastically reduce the time needed to verify the products of an SPL, they may still suffer from the state explosion problem. Overcoming this issue is a well-known challenge in explicit state space model checking. Symbolic algorithms, which make use of symbolic representations of the state space, are a solution to this problem. They have shown to be particularly efficient in the context of single-systems model checking and made possible the verification of huge systems [34].

Moreover, FTSs is a foundational formalism, not meant to be used directly by engineers. It is thus important to relate the FTS language to high-level languages that can be used in industrial settings. A suitable language for SPLs is *fSMV*, which was introduced by Plath and Ryan [36]. fSMV extends SMV (i.e. the (Nu)SMV model checker's input language) with primitives that allow to account for the addition of new features. More precisely, in fSMV, an SPL is represented by a base system described in the SMV language. Each additional feature is described independently, stating its assumptions and modifications. A product is built from the base system by adding features in a certain order.

In [36], Plath and Ryan propose a procedure to verify properties of a product. The verification procedure exploits the fact that a product can be expressed with SMV alone, and hence semantically as a *transition system*. This property allows them to reuse the classical symbolic verification procedure implemented in the (Nu)SMV toolset, which provides an efficient way to verify *a single product*. An fSMV model represents several products, each being the combination of the base system and a set of features. However, with the approach described in [36], one check per product is needed, which decreases the performance of the approach considerably. This is because transition systems do not allow to distinguish between features and hence between products. There is thus a need for a translation from fSMV to a formal model that is more suited to represent SPLs.

The contribution of the present paper is twofold. First, we propose symbolic algorithms for model checking an FTS against temporal properties. Second, we study the relation between FTSs and fSMV and provide them with a bi-directional translation. For this purpose, we provide the fSMV language with a definition different from the one given by Plath and Ryan [36]. The formal definition of fSMV is different from the one of SMV as we now have to characterise features and feature composition. Then, we show that FTSs and fSMV are equally expressive, that is, any FTS can be translated into an equivalent (in terms of behaviour) fSMV model and vice versa. This proof provides evidence that fSMV is an appropriate notation to model SPL behaviour. It is also a reference against which implementations can be proven correct, which is necessary to obtain trustworthy verification tools. Finally, we exploit this result to extend the NuSMV model checker with