Science of Computer Programming 168 (2018) 171–185

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Basic behavioral models for software product lines: Revisited

Mahsa Varshosaz [a,*], Harsh Beohar [b], Mohammad Reza Mousavi [c]

[a] Center for Research on Embedded Systems (CERES), Halmstad University, Sweden
[b] University of Duisburg-Essen, Germany
[c] University of Leicester, UK

ARTICLE INFO

Article history:
Received 28 May 2017
Received in revised form 2 September 2018
Accepted 3 September 2018
Available online 7 September 2018

Keywords:
Software product lines
Behavioral model
Featured transition systems
Calculus of communicating systems
Product line labeled transition systems

ABSTRACT

In Beohar et al. (2016) [9], we established an expressiveness hierarchy and studied the notions of refinement and testing for three fundamental behavioral models for software product lines. These models were featured transition systems, product line labeled transition systems, and modal transition systems. It turns out that our definition of product line labeled transition systems is more restrictive than the one introduced by Gruler, Leucker, and Scheidemann. Adopting the original and more liberal notion changes the expressiveness results, as we demonstrate in this paper. Namely, we show that the original notion of product line labeled transition systems and featured transition systems are equally expressive. As an additional result, we show that there are featured transition systems for which the size of the corresponding product line labeled transition system, resulting from any sound encoding, is exponentially larger than the size of the original model. Furthermore, we show that each product line labeled transition system can be encoded into a featured transition system, such that the size of featured transition system is linear in terms of the size of the corresponding model. To summarize, featured transition systems are equally expressive as, but exponentially more succinct than, product line labeled transition systems.

1. Introduction

Software Product Line (SPL) engineering is a software development technique enabling mass production and customization. Using this technique, a family of software systems is efficiently developed based on a common core and by benefiting from systematic reuse of artifacts among products.

There are several quality assurance techniques such as model-based testing and model checking that require a model describing the behavior of the system. Hence, several behavioral models have been proposed that can be used for compactly and efficiently representing the behavior of the products in an SPL; examples of such models are Featured Transition Systems (FTSs) [1], Product Line Calculus of Communicating Systems (PL-CCSs) [2], and Modal Transition Systems (MTSs) [3] and different extensions of MTSs [4–7]. These formalisms are comparable in terms of the types of behavior that they can capture and also in terms of their underlying formal model, i.e., Labeled Transition Systems (LTSs).

FTSs [1] are introduced as an extension of LTSs where the transitions are additionally labeled with feature expressions. Each feature expression is a propositional formula in which the variables represent the features of a product family. Feature expressions indicate the presence/absence of a transition in the model of each product (for more details see Section 2.2).

* Corresponding author.
E-mail addresses: mahsa.varshosaz@hh.se (M. Varshosaz), harsh.beohar@uni-due.de (H. Beohar), mm789@le.ac.uk (M.R. Mousavi).

https://doi.org/10.1016/j.scico.2018.09.001
0167-6423/

Using FTSs, the behavior of all products is represented in a whole model and different types of analysis can be performed for all products at once using this model.

PL-CCSs [2] are an extension of Milner's Calculus of Communicating Systems (CCSs) [8]. Using PL-CCSs, it is possible to model alternative behavior. The syntax of PL-CCS is an extension of the syntax of CCS with a *variant* operator, which represents an alternative choice between its operands. A choice can be resolved once and for all. This means, in case of recursion, that if a variant choice is resolved in the first iteration, then it remains the same in the future iterations. In [2], Product Line Labeled Transition Systems (PL-LTSs) are defined as the semantic domain for PL-CCS models. In order to keep track of variant choices, a configuration vector is included in the state of PL-LTSs. In each PL-LTS, the size of the vector is equal to the number of variant choices in the corresponding PL-CCS term. The elements of the configuration vector can denote a choice that is either undecided or decided in favor of the left-hand side or right-hand side variant.

In [9], we studied the comparative expressiveness of three of the above formalisms, namely FTSs, PL-CCSs and MTSs, where as a part of the results, we concluded that the class of PL-LTSs is less expressive than the class of FTSs (see Theorem 4 in [9]). In this work, it was assumed that in PL-LTSs in each step, only one of the variant choices can be resolved. Based on this assumption each transition can change only one of the elements of the configuration vector in the target state. This turns out to be an overly restrictive assumption compared to the definition given for the PL-LTS transition rules in [2]. Considering this assumption, it was shown that PL-LTSs cannot capture some types of behavior such as three-way choices which can be captured by FTSs.

In this paper, we relax the above-mentioned restriction and adapt the result to the original and more liberal definition of PL-LTSs [2]. We revisit the comparative expressiveness of FTSs and PL-LTSs with respect to the products that they specify. We describe an encoding of FTSs into PL-LTSs and there by proving that for each FTS, the set of LTSs that implement the FTS are also valid implementations for the PL-LTS resulting from the encoding. The results show that the class of PL-LTSs is at least as expressive as the class of FTSs. We also show that the results provided in [9], specifying that the class of FTSs is at least as expressive as the class of PL-LTSs still holds. Hence, we conclude that the class of PL-LTSs and the class of FTSs are equally expressive. We also provide a comparative succinctness analysis of the size of the PL-LTSs resulting from any sound encoding in terms of the number of states of the corresponding FTS. The results of the succinctness analysis show that FTSs are more succinct formalisms compared to PL-LTSs to describe SPLs.

The rest of this paper is organized as follows. In Section 2, we review the basic definitions regarding FTSs and PL-CCSs. In Section 3, we provide encodings between FTSs and PL-LTSs. In Section 4, we show that the class of PL-LTSs, i.e., underlying semantic model of PL-CCSs, and the class of FTSs are equally expressive. In Section 5, we provide a comparative succinctness analysis for the models resulting from encoding of FTSs. In Section 6, we conclude the paper and present the directions of our ongoing and future work.

2. Preliminaries

In this section, we provide the definition of constructs and concepts that are used throughout the paper.

2.1. Feature diagram

In SPL engineering, the commonalities and variabilities among products are described using *features*. A *feature* is defined as "a prominent or distinctive user-visible aspect, quality, or characteristic of a software system" [10]. Each product in an SPL is defined by a subset of features selected from the whole set of features of the SPL. There are different relations between the features in an SPL. *Feature models* [11] are a common means to compactly represent the set of products of an SPL in terms of its features.

A feature model is a hierarchical structure consisting of nodes and edges between them. Each node in a feature model represents a feature in the SPL. The structure of a feature model is tree like. Each node can have a set of child nodes. The features in an SPL can be *optional*, or *mandatory*. The mandatory features are present in all products of the SPL, while the optional features may be present in a subset of the products. A group of sibling features (nodes) can have the *alternative* relation, which means only one of the features in the group can be included in a product in case that the parent feature is selected. Also, a group of sibling features can have the *or* relation, which means one or more features in the group can be included in a product if the parent feature is selected. There are cross tree relations such as *requires* (resp. *excludes*), where the inclusion of a feature results in inclusion (resp. exclusion) of other features. Each feature model can be represented by a propositional logic formula in which propositional variables represent the features in the SPL [12].

Example 1. An example of a feature model is depicted in Fig. 1. The feature model corresponds to a vending machine product line (the vending machine in this example is a simplified version of the one given in [9]).

In this feature model features such as *coin* (o), *beverage* (b), and *coffee* (c) are mandatory and features *tea* (t) and *cappuccino* (p) are optional. (The single letters given under each feature are used later to represent the features in the propositional formulae.) The set of features *coffee* (c), *cappuccino* (p) and *tea* (t) have the *or* relation. Also, features *1e* (e) and *1d* (d) have the *alternative* relation, which means the machine can take only one type of coin (euro or dollar). The dashed two headed arrow represents the *excludes* relation between the *cappuccino* (p) and the 1*d* (d) features.