Science of Computer Programming 83 (2014) 35–55

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Reducing the verification cost of evolving product families using static analysis techniques[☆]

Hamideh Sabouri[a,*], Ramtin Khosravi[a,b,*]

[a] School of Electrical and Computer Engineering, University of Tehran, Karegar Ave., Tehran, Iran
[b] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran

HIGHLIGHTS

- We model product families using Rebeca.
- We reduce the number of verified products.
- We re-use previous result of verification after evolution of product lines.
- The results show the effectiveness of our approach.

ARTICLE INFO

Article history:
Received 3 May 2012
Received in revised form 19 June 2013
Accepted 21 June 2013
Available online 19 July 2013

Keywords:
Software product lines
Model checking
Program slicing
Static analysis
Reduction techniques

ABSTRACT

Software product line engineering enables proactive reuse among a set of related products through explicit modeling of commonalities and differences among them. Software product lines are intended to be used in a long period of time. As a result, they evolve over time, due to the changes in the requirements. Having several individual products in a software family, verification of the entire family may take a considerable effort. In this paper we aim to decrease this cost by reducing the number of verified products using static analysis techniques. Furthermore, to reduce model checking costs after product line evolution, we restrict the number of products that should be re-verified by reusing the previous verification result. All proposed techniques are based on static analysis of the product family model with respect to the property and can be automated. To show the effectiveness of these techniques we apply them on a set of case studies and present the results.

1. Introduction

Software product line (SPL) paradigm enables proactive reuse by developing a family of related products instead of developing each product separately. To this end, the commonalities and differences among products should be modeled explicitly [1]. Feature models [2] are widely used for this purpose. A feature model is a tree of features, containing mandatory and optional features as well as a number of constraints among them. A product is then defined by a combination of features, and product family is the set containing all valid feature combinations.

Software product line engineering is used in the development of embedded and critical systems [3]. Therefore, formal modeling and verification of software product families is essential for these types of applications. To ensure that certain properties hold in product families, we can apply model checking [4] which is a promising technique to develop more reliable systems. However, it has high computational and memory cost which may lead to the well-known problem of state

[☆] This research was in part supported by a grant form IPM (No. CS1390-4-02).
[*] Corresponding authors.
E-mail address: hamideh.sabouri@gmail.com (H. Sabouri).

0167-6423/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.scico.2013.06.009

space explosion. This problem limits the applicability of this technique to verify product families where the number of products may be exponential in the number of features.

The main purpose of our work is to decrease the cost (i.e. memory usage and time) of model checking and tackle the state space explosion problem by reducing the number of products that need to be verified. To this end, we analyze a product family model statically, before verifying it against a given property. Our work centers around the notion of program slicing [5]. Program slicing is a static analysis technique that extracts the statements from a program that are relevant to a particular computation. One of the main approaches for slicing is using reachability analysis on the program dependence graph [6]. The nodes of a program dependence graph are the statements of the program and its edges represent data and control dependencies among the statements. In this paper, we adapt the program dependence graph and the reachability algorithm to:

– Detect the features whose presence does not affect satisfaction of a given property. For such features, it suffices to verify the products that exclude those features.
– Determine satisfaction/violation of a given property for a subset of products statically (without verification).
– Among the set of alternative features (in the feature model), identify those sets for which the specific choice of feature does not affect satisfaction/violation of the property. We only verify products that include one of those features and exclude the others.

Software product lines are intended to be used in a long period of time. As a result, they evolve over time due to changes in the requirements [7]. The evolution may imply addition, removal, or replacement of some features or it may add or remove some constraints in the feature model. Moreover, a statement may be added to or modified in the product family model. To decrease the cost of model checking, we do not re-verify the entire product family after evolution. Instead, we statically analyze the model to:

– Determine the products that belong to the product line before and after evolution. Satisfaction/violation of a property in these products can be derived from the results of verification before evolution.
– Investigate the reachability of the added/modified statement in different products. We only re-verify the products where the statement is reachable.
– Identify the properties affected by the added/modified statement. We do not re-verify the properties that are not affected by that statement.

To justify the applicability of the proposed techniques, we discuss the computation cost of applying each technique. We evaluate our proposed approach by applying it on a set of case studies. Then, we discuss the effectiveness of the techniques according to the obtained results.

This paper modifies and extends our previous paper [8] by (1) improving our technique to determine satisfaction/violation of a property statically for a larger set of products, (2) proposing a technique to identify alternative features where the specific choice for the alternative does not affect satisfaction/violation of the property, (3) presenting three techniques to reduce the verification cost after product line evolution, (4) discussing the computation cost of the proposed techniques, (5) enriching the evaluation of our approach by applying it on several case studies, and (6) discussing the effectiveness of the proposed technique.

The coffee machine example. We use the coffee machine family as the running example in this paper. A coffee machine may serve coffee, tea, or both. Adding extra milk and extra sugar may be supported optionally. The payment method of a coffee machine is either by coin or by card.

This paper is structured as follows. Section 2 discusses the related work. In Section 3, feature models are described. In Section 4, we explain our modeling and model checking approach to verify product families. In Section 5, we elaborate our proposed techniques to reduce the number of verified products followed by Section 6 where we explain how to decrease model checking cost of evolved product lines. Section 7 presents a discussion on complexity of the proposed techniques. In Section 8, we describe the case studies and depict the effect of the reduction techniques on the number of states and time of verification. Finally, Section 9 concludes the work.

2. Related work

2.1. Modeling and model checking SPLs

Recently, several approaches have been developed for formal modeling of product lines using transition systems [9–13], process algebra [14], Petri-nets [15], and ABS modeling language [16]. These approaches capture the behavior of the entire product family in a single model by including the variability information in it. In such a model, it is specified how the behavior changes when a feature is included or excluded.

Model checking of product lines is discussed in [11–14,17]. In these approaches, the model checker investigates all of the possible feature combinations when verifying the model of a product family against a property. The result of model checking is the set of products that satisfy the property. The focus of these works is on adapting model checking algorithms