Information and Software Technology 56 (2014) 1122–1143

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Efficient synthesis of feature models

Steven She [a], Uwe Ryssel [b], Nele Andersen [c], Andrzej Wąsowski [d,*], Krzysztof Czarnecki [a]

[a] Generative Software Development Lab, Department of Electrical and Computer Engineering, University of Waterloo 200 University Avenue West Waterloo, Ontario, N2L 3G1, Canada
[b] Technische Universität Dresden, Fakultät Informatik, Institut für Angewandte Informatik, D-01062 Dresden, Germany
[c] Configit A/S, Kristianiagade 7, 2100 Copenhagen, Denmark
[d] IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen, Denmark

ARTICLE INFO

Article history:
Available online 3 February 2014

Keywords:
Feature models
Variability models
Software product lines

ABSTRACT

Context: Variability modeling, and in particular feature modeling, is a central element of model-driven software product line architectures. Such architectures often emerge from legacy code, but, creating feature models from large, legacy systems is a long and arduous task. We describe three synthesis scenarios that can benefit from the algorithms in this paper.
Objective: This paper addresses the problem of automatic synthesis of feature models from propositional constraints. We show that the decision version of the problem is NP-hard. We designed two efficient algorithms for synthesis of feature models from CNF and DNF formulas respectively.
Method: We performed an experimental evaluation of the algorithms against a binary decision diagram (BDD)-based approach and a formal concept analysis (FCA)-based approach using models derived from realistic models.
Results: Our evaluation shows a 10 to 1,000-fold performance improvement for our algorithms over the BDD-based approach. The performance of the DNF-based algorithm was similar to the FCA-based approach, with advantages for both techniques. We identified input properties that affect the runtimes of the CNF- and DNF-based algorithms.
Conclusions: Our algorithms are the first known techniques that are efficient enough to be used on dependencies extracted from real systems, opening new possibilities of creating reverse engineering and model management tools for variability models.

1. Introduction

Variability models are central to development and management of software product lines (SPL) and comprise of simple *problem space models* and typically complex *solution space models*. A problem space model describes major decisions made during customization—such as whether an Enterprise Resource Planning (ERP) system should include an e-commerce platform or not. The solution space model explains how the problem space decisions affect the realization. For example, how the e-commerce platform is woven into the implementation, by extending data models, user interfaces and services.

Variability models contain concepts referred to as decisions [1], features [2] or variation points [3], depending on the abstraction level. The abstract models tend to contain relatively few concepts (up to hundreds in the largest models[1]), while the low level concrete models can reach thousands of variation points. These concepts are typically organized hierarchically, and related to each other using constraints. There exist multiple commercial (Pure Systems GmbH, Big Lever Software Inc.) and research [4–6,1] tools for variability modeling. Recognizing the increasing significance of this market segment, The Object Management Group (OMG) has initiated [7] a standardization process for the Common Variability Language (CVL).

Feature models [2,8] are a prominent notation used in variability modeling. Applications of feature modeling include automatic generation of product configurators, driving code generators [8] and build systems [9] to compose individual members of an SPL, and driving test and verification [10,11]. Feature models will also be part of the CVL standard [7]. In this paper, we use the term feature in the abstract unifying sense, meaning either a decision or a variation point. This simplification is justified, since we will be exploiting the combinatorial structure of features, which is similar

* Corresponding author. Tel.: +45 7218 5086.
E-mail addresses: shshe@gsd.uwaterloo.ca (S. She), uwe.ryssel@tu-dresden.de (U. Ryssel), nele.andersen@gmail.com (N. Andersen), wasowski@itu.dk (A. Wąsowski), kczarnec@gsd.uwaterloo.ca (K. Czarnecki).

[1] Based on personal communications with Big Lever and Pure Systems.

http://dx.doi.org/10.1016/j.infsof.2014.01.012
0950-5849/

in both the solution space and in the problem space.

We show a feature model for a power management subsystem in Fig. 1. The feature diagram is composed of rectangles or features, and lines connecting a parent feature to its children. An empty circle indicates an *optional* feature, and a filled circle indicates a *mandatory* feature. An empty arc between two or more features denotes an xor-group where exactly one of its features must be selected. Feature diagrams can also support or-groups—where one or more features must be selected, or mutex-groups—where at most one feature must be selected. Additional cross-tree constraints can be specified as part of an arbitrary cross-tree formula below the diagram, or as edges for either implies or excludes constraints.

SPLs are typically large software projects that often result from a long lasting evolution, based on substantial legacy code. Industrial SPLs use models containing thousands of features that mix both the problem and the solution spaces. For instance, the variability in the Linux kernel has more than 5000 features that describe its x86 architecture [9]. At the same time, other SPLs, such as the FreeBSD kernel, do not use any variability models and could benefit from having a feature model. This situation and scenario also arises in industry as we found in our communications with Pure Systems, the company behind the pure::variants tool.

Reverse engineering techniques for variability models would ease adoption of product line practices. They enable a smoother migration of legacy code to systematic product line architectures and their subsequent evolution. This paper addresses the problem of feature model synthesis, which is the core algorithmic part of reverse engineering. Given a set of features and dependencies, our algorithms constructs feature diagrams that contain a hierarchy of features, enriched by cross-hierarchy implies and excludes constraints. Our algorithms assume that the input is expressed in propositional logics. In practice, the input dependencies can be either specified by engineers, or automatically mined from the source code using static analysis [12]. Furthermore, effective management of large feature models requires model management operations such as merge, compare, diff, and project [13]. Such operations ease model evolution by allowing developers to compare models to assess the impact of model edits or build large models by composing smaller ones. The feature model synthesis problem is also core to these model operations that are defined via logical operators on formulas derived from the input models [13]. We describe these scenarios further in Section 2.

In this paper, we formally define the problem of synthesis of feature models, discuss its complexity, derive semantics-based algorithms and argue for their correctness. Technically, we synthesize not a feature model, but a *feature graph*, which is a symbolic representation of all possible feature models that could be sound results of the synthesis. Then, we show that any of these models can be efficiently derived from the feature graph. Our contributions include:

- Definition of feature model synthesis as an algorithmic problem, an NP-hardness result, and a complexity driven analysis of suitable solution techniques.
- An algorithm for synthesis of feature models from *conjunctive normal form* (CNF) formulas that is at least 10-times faster than previously known algorithms based on our presented performance evaluation.
- An efficient algorithm for synthesis of feature models from *disjunctive normal form* (DNF) formulas.
- An evaluation of our algorithms using a dataset derived from realistic feature models and a dataset of generated models. This evaluation includes a comparison of the CNF-based algorithm against an existing BDD-based algorithm [14], and a comparison of the DNF-based algorithm against an FCA-based algorithm [15]. We discuss factors for predicting the runtime of our algorithms.

The above techniques produce feature models, but can be easily adjusted to other languages, such as the propositional part of variability specifications of the current CVL proposal. A variability specification in CVL is essentially a cardinality-based feature model and we can reuse our techniques for feature modeling to synthesize these trees. More importantly, the algorithm for synthesis of feature models from a constraint in CNF form is the first known technique for this problem. These synthesis algorithms can be applied to data extracted from real systems. The previous work of the same authors [14] has shed light on the mathematical structure of the problem, but has failed to provide scalable algorithms.

We first analyze computational complexity of the individual steps in the synthesis of feature models and of variations of the problems for different input representations (i.e., formulas in CNF and DNF). The complexity analysis allows us to decide what the promising reductions of the problem are; for example using SAT-based techniques for synthesis of or-groups from DNF formulas, and not using these techniques for CNF formulas. We exploit this in the design of CNF and DNF algorithms that we describe in the following sections.

1.1. Extensions from our previous contribution

The previous paper on the subject [16] introduced the FGE-CNF and FGE-DNF algorithms for synthesizing feature graphs from formulas in CNF and DNF respectively. We expand the previous paper with the following contributions:

- Extended Section 2 with additional workflows and scenarios from [17].
- Expanded the evaluation dataset for both FGE-CNF and FGE-DNF with 267 models from the SPLOT model repository [18].
- Expanded description of the BDD-based implementation from [14], and the formal concept analysis-based implementation by Ryssel et al. [15] according to the FGE algorithm in Fig. 7.
- A comprehensive evaluation of FGE-DNF against the formal concept analysis-based algorithm by Ryssel et al. [15]. We identify differences between the two algorithms and compare the runtimes of the two algorithms in Section 8.

2. Scenarios and motivation

In this section, we describe several scenarios involving feature model synthesis. We describe the entire workflow from the input artifacts, to how they are transformed into the inputs used by our synthesis algorithms. While feature model synthesis is only

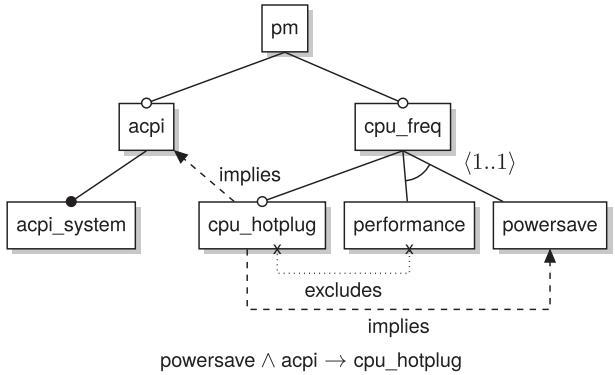

Fig. 1. Feature model of a power management subsystem.