Journal of Logical and Algebraic Methods in Programming 97 (2018) 30–54

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

A theoretical framework for cardinality-based feature models: The semantics and computational aspects [☆]

Aliakbar Safilian [*], Tom Maibaum, Zinovy Diskin

Department of Computing and Software, McMaster University, Canada

ARTICLE INFO

Article history:
Received 26 June 2017
Received in revised form 22 February 2018
Accepted 24 February 2018
Available online xxxx

Keywords:
Cardinality-based feature models
Formal language theory
Decidability
Complexity

ABSTRACT

Feature modeling is the most common approach for modeling software product line configurations. We propose a formal language-based formalization for the hierarchical semantics of cardinality-based feature models. We provide a transformation mapping, which allows us to transform a cardinality-based feature diagram to an appropriate regular expression. We propose a formal framework for expressing crosscutting constraints over cardinality-based feature diagrams. We then provide two kinds of semantics for constraints: the flat and the language semantics. We show how to integrate the semantics of diagrams and constraints over them. We also characterize some analysis operations over feature models in terms of operations on languages and discuss the corresponding decidability and computational complexity problems.

1. Introduction

Product line (PL) engineering [5] is a very well-known industrial approach to software/hardware design. A PL is a set of products that share some *commonalities* and exhibit some *variabilities*, where commonalities and variabilities are usually captured using entities called *features*, "system properties that are relevant to some stakeholders" [16]. The idea of this approach is that, instead of producing products individually, the common core of a PL is first produced, leaving a much smaller task to be completed, namely the adaptation of the core to a concrete application requirement. This results in several advantages, including a significant reduction in development time/cost and an increase in reusability [5].

The most common method for modeling PLs is *Feature modeling*. A *feature model* (FM) is a tree of features representing a hierarchical structure of features, which is equipped with some special annotations on the tree's elements showing the *constraints* on features, based on which valid configurations are built. An FM may also be equipped with some additional constraints called *crosscutting constraints* (CCs), a.k.a. *cross-tree constraints* [19]. As the name suggests, these constraints are defined on *incomparable* features. Two features are called incomparable if neither of them is a descendant of the other in the hierarchical structure. We distinguish between these constraints and the tree-structure in a feature model, and call the latter a *feature diagram*.[1]

Feature modeling languages could be divided into *non-attributed* (we call *pure*) and *attributed* FMs. Pure FMs are grouped into *Boolean* and *cardinality-based* FMs. Boolean FMs [28] represent product variability and commonality in terms of Boolean

☆ Financial support was provided by APC via the NECSIS project.

* Corresponding author.
E-mail addresses: safiliaa@mcmaster.ca (A. Safilian), maibaum@mcmaster.ca (T. Maibaum), diskinz@mcmaster.ca (Z. Diskin).

[1] There are some feature diagram variants that represent some simple crosscutting constrains (*inclusive* and *exclusive* constraints involving only two features) [29].

https://doi.org/10.1016/j.jlamp.2018.02.002
2352-2208/© 2018 Elsevier Inc. All rights reserved.

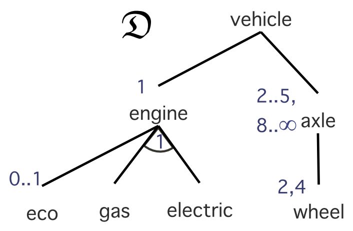

Fig. 1. A cardinality-based feature model for vehicles.

constraints: *optional/mandatory* features, *OR/XOR* decomposition operations, and Boolean CCs. These models are not expressive enough to model PLs that deal not only with the feature types, but also with the number of feature instances. In cardinality-based FMs (CFMs) [16], multiplicity constraints are used to represent the constraints of PLs' configurations. Features in attributed FMs [49] are equipped with some attributes allowing them to model extra quantitative constraints. In this paper, we restrict ourselves to CFMs. Since CFMs subsume Boolean FMs [16],[2] the research reported in this paper can be also applied to Boolean FMs.

Fig. 1 represents a cardinality-based feature diagram (CFD) for a vehicle. A vehicle must have exactly one engine and any number of axles (imagine a futuristic vehicle model), excluding zero, one, six, and seven. The multiplicity constraints "1" and "$2..5, 8..\infty$" on engine and axle, respectively, model these requirements. An engine can be optionally equipped with an ecosystem, represented by the subfeature eco with multiplicity constraint "0..1". Arcs on a set of sibling features equipped with multiplicity constraints model *groups*. An engine can be either gasoline or electric but not both (note the multiplicity constraint "1" on the group $\{\mathsf{electric}, \mathsf{gas}\}$). An axle has either two or four wheels (note the multiplicity constraint $2, 4$ on wheel). This CFD, denoted by $\mathfrak{D}$, is used as a running example throughout the paper. Suppose that our model also needs to satisfy the following requirement: "if an engine is equipped with an ecosystem, then the number of axles must be 4". This cannot be expressed by multiplicity constraints on the diagram's elements and should be considered as a CC. Let us denote the whole CFM (i.e., $\mathfrak{D}$ and the above CC) by $\mathfrak{M}$.

The most common semantics considered for a CFM in the literature is the set of its valid *flat* configurations, where a flat configuration of a CFM is a multiset of features satisfying the constraints of the CFM [16]. As an example, the multiset of features $\{\{\mathsf{vehicle}, \mathsf{engine}, \mathsf{electric}, \mathsf{eco}, \mathsf{axle}^2, \mathsf{wheel}^6\}\}$ is a valid flat configuration of the CFM $\mathfrak{M}$ above.[3] The set of all flat configurations is called the *flat semantics* of the CFM [42].

The flat semantics of a given CFM ignores the hierarchical structure of the CFM. Capturing hierarchical structures of CFMs is important for several reasons: Some analysis questions about the CFM rely explicitly on the hierarchical structure of the CFM, including the least common ancestor (LCA) of a given set of features and determining the root feature [34]. Such analysis questions can be used for extracting some important information of the CFM. For example, the LCA operation comes in handy when one wants to get a smallest subsystem of a system (modeled by an FM) such that it includes some specific properties (features). Some other operations, *specialization* and *refactoring* [50], compare two (or more) given FMs in a semantic sense.[4] Relying on a poor abstraction (like the flat semantics) to define these analysis operations would make their definitions deficient. For instance, consider two very simple Boolean FMs $\mathbf{M}_1$ and $\mathbf{M}_2$ that are defined on the same set of features $\{a, b, c\}$ as follows: All the features in both models are mandatory and their tree-structure are represented by "$a = b^{\uparrow} = c^{\uparrow}$" and "$a = b^{\uparrow}, b = c^{\uparrow}$", respectively. ($f = g^{\uparrow}$ denotes that f is the parent of g.) They are equivalent in the flat semantics. Nevertheless, they represent two different PLs, as c is a constituent of b in the latter while this is not the case in the former. There are also some other challenging tasks in the domain of feature modeling, including *reverse engineering* of FMs and *feature model management*, in which preserving the hierarchical structure of given FMs is essential: It is obvious why preserving the hierarchical structure of FMs is important in reverse engineering, as the output of a reverse engineering task must be (ideally) an FM. Given two FMs, the output of a model management operation (e.g., merge) should satisfy some invariant properties derived from the given models, including the hierarchical structures of the FMs. In light of the above discussion, we define a *faithful semantics* of a CFM as a semantics capturing both the flat semantics and the hierarchical structure of the CFM.

The main contributions of the paper are summarized as follows:

– We provide a faithful semantics for CFDs by using regular languages as the semantic domain. Because of computational properties of regular languages (e.g., their closure properties and complexity class SPACE(O(1))), we can claim that this transformation provides an efficient computational framework for reasoning about CFDs.

– We then propose a formal abstract framework for expressing CCs over CFDs. We provide two kinds of semantics for CCs, flat and language-based (using the class of context-sensitive languages as the semantic domain).

– We characterize some existing analysis operations over CFMs in terms of corresponding operations over languages and discuss their computational aspects, i.e., the corresponding decidability and complexity problems.

[2] This is because any Boolean constraint can be expressed in terms of multiplicities.

[3] We use the notations "{{" and "}}" to identify multisets – for a precise definition, please see Sect. 2.1.

[4] The former investigates their equality and the latter investigates their subsetting relation.