Feature-Model Interfaces for Compositional Analyses

Reimar Schröter [*], Sebastian Krieter [*], Thomas Thüm [†], Fabian Benduhn [*], Gunter Saake [*]
[*] University of Magdeburg, [†] Technische Universität Braunschweig
Germany

ABSTRACT

Feature models are often used to describe the commonality and variability in a software product line. A feature model describes the possible combinations of features for generation of products. To ensure the correctness of feature models, several authors propose automated analyses (e.g., to guarantee that at least one product can be generated). However, industrial product lines contain thousands of features and, thus, the specification of a feature model as well as the product configuration becomes challenging. We uncover that existing modularization techniques for feature models are not sufficient for compositional analyses. In this paper, we present the concept of feature-model interfaces to close this gap. We prove that feature-model interfaces can be used to modularize feature models in a way that supports compositional analyses.

Keywords

Software Product Lines, Variability Modeling, Feature Models, Modularity, Compositional Analysis

1. INTRODUCTION

Feature models are often used to describe the commonality and variability in a software product line [5]. The common and variable artifacts are represented by features, which are arranged in a tree structure with additional cross-tree constraints to describe all valid feature combinations of the product line [6, 13]. Since cross-tree constraints can be arbitrary propositional formulas, a feature model may be inconsistent (e.g., the feature model involves *dead features* that do not exist in any product). Therefore, developers have to check a feature model for existing inconsistencies and unintentional dependencies. *Automated analyses* were proposed to efficiently detect such problems in feature models [7].

Feature models can be very large; our industrial partners face the challenge of feature models with more than 10 000 features. However, the general concept of feature-model analyses does not scale for large feature models regarding all analysis types (e.g., SAT-based analysis to determine the *number of products*). Therefore, large feature models lead to problems regarding manual comprehension and automated analyses. Several approaches were proposed that divide feature models into smaller parts to improve comprehensibility and to reduce the complexity. Furthermore, current techniques allow us to combine these parts in a flexible manner to build new feature models [1, 4, 20]. However, these composition techniques do not support compositional analyses, i.e., they still rely on analyses of the composed model that includes all information of all involved submodels.

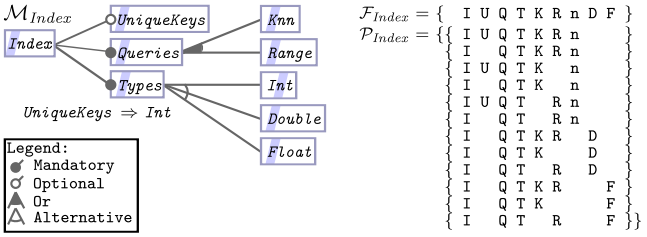

Figure 1: Feature model $\mathcal{M}_{Index} = (\mathcal{F}_{Index}, \mathcal{P}_{Index})$.

We propose to use feature-model interfaces to compose feature models in a way that supports compositional analyses. Feature-model interfaces are a special kind of feature models to supports the concept of *information hiding* in the feature-modeling process [14]. Thus, the feature-model interface only consists of features that a developer is intended to use in a specific composition scenario, whereas all other features are hidden to the developer. Therefore, feature-model interfaces reduce the complexity of the composed model and we will show that feature-model interfaces can also be used to apply compositional analyses.

In particular, we make the following contributions:

- We formally define feature-model interfaces based on a definition of feature models.
- We discuss how to benefit from feature-model interfaces in feature-model compositions.
- We prove the correctness of using feature-model interfaces for compositional feature-model analyses.

2. FEATURE-MODEL ANALYSES

In this section, we give a brief introduction on feature models and their automated analyses. A feature model consists of a set of features that are arranged in a tree structure with additional cross-tree constraints to describe all valid feature combinations [6, 13]. According to the tree structure, it is forced that in each product in which a child feature is included, the parent is also included. Besides this dependency, several kinds of child features exist: grouped features, *mandatory*, and *optional* features. *Mandatory* features are included in each product in which their parent is included, whereas the inclusion of an *optional* feature is not required.

Grouped features can be arranged in an *alternative-group* as well as in an *or-group*. While the *alternative-group* forces the existence of exactly one grouped feature if their parent is included, the *or-group* forces the existence of at least one feature. Beside the tree structure, it is possible to add cross-tree constraints (i.e., propositional formulas over the set of features) to reduce the represented set of products.

In Figure 1, we present the feature model *Index*, which represents a set of index structures to optimally support direct access of data items in a database. An index can only support one data-type at a time. Thus, the features *Int*, *Double*, and *Float* are arranged in an alternative-group. Furthermore, the developer can choose the query algorithms *Knn*, and *Range* to search for data items. The query algorithms are independent of each other and are therefore represented by an or-group. Furthermore, it is optional to force unique keys in an index structure for which we include an optional feature *UniqueKeys*. Since it is only possible to support unique keys for integer values, the model contains the additional cross-tree constraint *UniqueKeys* $\Rightarrow$ *Int* as propositional formula.

For our proofs, we define feature models as follows:

DEFINITION 1. *A feature model $\mathcal{M}_x$ is a tuple $(\mathcal{F}_x, \mathcal{P}_x)$, where:*

- *$\mathcal{F}_x$ is a set of features, and*
- *$\mathcal{P}_x$ is a set of products with $\mathcal{P}_x \subseteq 2^{\mathcal{F}_x}$.*

A feature model is specified by a set of features and all combinations of features that lead to valid products.

Let us take a look at our example of feature model *Index*. Formally, we can define this feature model as $\mathcal{M}_{Index} = (\mathcal{F}_{Index}, \mathcal{P}_{Index})$. In Figure 1, we exemplify the sets $\mathcal{F}_{Index}$ and $\mathcal{P}_{Index}$ on the right side using the highlighted characters of the graphical representation as feature diagram.

Automated Analyses of Feature Models

In industry, feature models may consist of thousands of features (e.g., Linux kernel with more than 11 000 features [25]), which affects the comprehensibility of feature dependencies in a negative manner. Thus, automated consistency checks for large-scale feature models are of high importance. Benavides et al. present an overview of automated analyses and consider them as an information extraction process that is executed in two steps [7]. In the first step, an analysis tool translates the feature model into a specific representation (e.g., propositional logic). In the second step, a corresponding solver or algorithm is used to perform the analysis and to determine the analysis result. In the following, we present an excerpt of possible analyses that we investigate in the remainder of this paper and present a formal definition.

Void Feature Models. A feature model is void if and only if it represents no products [6, 7, 13]. Based on our definition of feature models, we can formalize the analysis of *void feature models* as follows. A feature model $\mathcal{M}_x$ is part of the set of all void feature models if and only if the set of products $\mathcal{P}_x$ is equal to the empty set.

$$void = \{\mathcal{M}_x \in \mathcal{M} \mid \mathcal{P}_x = \emptyset\}$$

Core Features. A *core feature* is a feature that is included in each product of the product line [7, 26]. Similar to the analysis of void feature models, we use our feature-model definition to formalize when a feature is a core feature. A feature is core if and only if it is an element of the set returned by the function *core*, which takes a feature model $\mathcal{M}_x$ as input.

$$core(\mathcal{M}_x) = \bigcap_{p \in \mathcal{P}_x} p$$

#	Features									Valid
	I	U	Q	T	K	R	n	D	F	
1	$\mathcal{F}_S$	$\mathcal{F}_D$	$\mathcal{F}_S$	$\mathcal{F}_S$	$\mathcal{F}_S$	$\mathcal{F}_D$	$\mathcal{F}_S$	$\mathcal{F}_D$	$\mathcal{F}_D$	✓
2	$\mathcal{F}_S$	$\mathcal{F}_D$	$\mathcal{F}_S$					$\mathcal{F}_S$		✓
3	$\mathcal{F}_S$						$\mathcal{F}_D$	$\mathcal{F}_D$		✗
4	$\mathcal{F}_S$	$\mathcal{F}_S$						$\mathcal{F}_S$		✗

Table 1: Example partial configurations of feature model $\mathcal{M}_{Index}$ (using highlighted characters of Figure 1 as abbreviations).

Dead Features. A feature of a feature model is a *dead feature* iff it is not part of any valid product of the product line [7, 13]. We use our feature-model definition to formalize the analysis of *dead features*. A feature is dead if and only if it is an element of the set returned by the function *dead*, which takes a feature model $\mathcal{M}_x$ as input.

$$dead(\mathcal{M}_x) = \mathcal{F}_x \setminus \bigcup_{p \in \mathcal{P}_x} p$$

Validity of Partial Configurations. A partial configuration is a tuple consisting of a set of selected features $\mathcal{F}_S$ and a set of deselected features $\mathcal{F}_D$. The validity analysis of partial configurations investigates whether a partial configuration contains a contradiction regarding the dependencies of a feature model [6, 7, 13]. Based on our feature-model definition, we define a function *vConf* that takes a feature model $\mathcal{M}_x$ as input and returns all existing valid partial configurations.

$$vConf(\mathcal{M}_x) = \{(\mathcal{F}_S, \mathcal{F}_D) \mid \exists p \in \mathcal{P}_x : \mathcal{F}_S \subseteq p \ \wedge \mathcal{F}_D \subseteq \mathcal{F}_x \setminus p\}$$

Example. Let us take a look at our feature model $\mathcal{M}_{Index}$ (see Figure 1). For the analysis of void feature models, we get the result that feature model $\mathcal{M}_{Index}$ is not in the set of all void feature models (i.e., $\mathcal{M}_{Index} \notin void$). If we analyze the core features, we get the set $core(\mathcal{M}_{Index}) = \{Index, Queries, Types\}$. By contrast, the application of function $dead(\mathcal{M}_{Index})$ results in an empty set and, thus, the feature model does not contain any dead features. In Table 1, we present four examples for partial configurations of feature model $\mathcal{M}_{Index}$. The partial configurations #1 and #2 are valid partial configurations. In addition, we call configuration #1 a complete configuration because each feature of $\mathcal{F}_{Index}$ is contained in either $\mathcal{F}_S$ or $\mathcal{F}_D$. By contrast, configurations #3 and #4 are invalid because they contain contradictions. In detail, the set $\mathcal{F}_D$ of configuration #3 contains two features of an alternative-group (*Int*, *Double*). In configuration #4 the features *UniqueKeys* and *Double* are selected at the same time, which is not possible due to

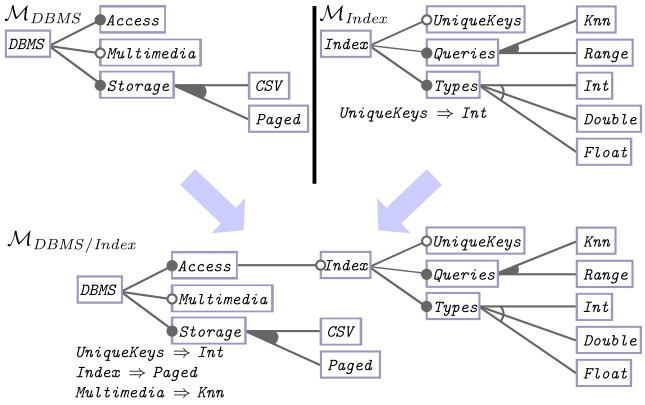

Figure 2: Aggregation of feature model $\mathcal{M}_{DBMS}$ and feature model $\mathcal{M}_{Index}$ with two additional cross-tree constraints.

the cross-tree constraint and alternative-group in the feature model.

3. FEATURE-MODEL COMPOSITION

To reduce the complexity of feature models and to improve their manageability, it is possible to divide feature models into smaller parts and describe the dependencies between them. For this, several composition mechanisms exist that allow us to combine feature models [4, 20]. In this paper, we consider the composition of feature models by means of aggregation, i.e., by inclusion of one feature model as an instance in another feature model [20].

Let us consider our initial example of the feature model $\mathcal{M}_{Index}$. We want to reuse an instance of the feature model $\mathcal{M}_{Index}$ in a database feature model ($\mathcal{M}_{DBMS}$). Since $\mathcal{M}_{Index}$ contains functionality that can be used to access items in the storage system of a database, it should be a child of the feature *Access* of $\mathcal{M}_{DBMS}$. We depict the intended result of this aggregation in Figure 2 as a new feature model $\mathcal{M}_{DBMS/Index}$ in which the root of $\mathcal{M}_{Index}$ (*Index*) is now a child of feature *Access*. Additionally, two other constraints are specified by domain experts and are added to the resulting feature model (see Figure 2).

In the remaining paper, we use aggregation to combine feature models and to perform automated analysis based on this composition. Therefore, we have to define the feature-model aggregation in a formal manner:

DEFINITION 2. *Let $\mathcal{M}_x = (\mathcal{F}_x, \mathcal{P}_x)$ and $\mathcal{M}_y = (\mathcal{F}_y, \mathcal{P}_y)$ and $\mathcal{M}_C = (\mathcal{F}_C, \mathcal{P}_C)$ be feature models with $\mathcal{F}_C \subseteq \mathcal{F}_x \cup \mathcal{F}_y$, we can describe the infix operator* aggregation $\circ_{\mathcal{M}_C}$ *of $\mathcal{M}_x, \mathcal{M}_y$ as follows:*

$$\circ_{\mathcal{M}_C} : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$$

$$\mathcal{M}_x \circ_{\mathcal{M}_C} \mathcal{M}_y = (\mathcal{M}_x \bullet R(\mathcal{M}_y)) \bullet \mathcal{M}_C \quad (1.1)$$

$$R : \mathcal{M} \to \mathcal{M}$$

$$R((\mathcal{F}_y, \mathcal{P}_y)) = (\mathcal{F}_y, \mathcal{P}_y \cup \{\emptyset\}) \quad (1.2)$$

$$\bullet : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$$

$$(\mathcal{F}_x, \mathcal{P}_x) \bullet (\mathcal{F}_y, \mathcal{P}_y) = (\mathcal{F}_z, \mathcal{P}_z) \quad (1.3)$$

$$= (\mathcal{F}_x \cup \mathcal{F}_y, \mathcal{P}_z) \quad (1.4)$$

$$= (\mathcal{F}_z, \{p \cup q \mid p \in \mathcal{P}_x, q \in \mathcal{P}_y, p \cap \mathcal{F}_y = q \cap \mathcal{F}_x\}) \quad (1.5)$$

The definition of the aggregation function $\circ_{\mathcal{M}_C}$ is based on two other functions $\bullet$ and R. Let us take a closer look at each of them.

Function R takes one feature model as input and converts it to a new feature model in which the empty product is a valid product. Thus, the feature set is identical to the input feature model and the set of products is extended by the empty set. R is used in function $\circ_{\mathcal{M}_C}$ to ensure that the root feature of $\mathcal{M}_y$ is not a core feature in the aggregated feature model.

Function $\bullet$ takes two feature models as input and returns a new combined feature model, which is a merge of all input information (i.e., features and products).[1] The resulting feature model consists of a set of features which is the union of all features from the input feature models. To combine the product sets of both input models, we use an operation that is similar to a join as known from the domain of databases. This operation combines the products of one feature model with the products of the other feature model. Like the join, we only combine two products if the additional condition $p \cap \mathcal{F}_y = q \cap \mathcal{F}_x$ is fulfilled.[2]

Function $\circ_{\mathcal{M}_C}$ represents our aggregation mechanism. The function uses two feature models as input and creates a new feature model by an application of the previously defined functions R and $\bullet$. The application of function $\bullet$ with $\mathcal{M}_x$ and $R(\mathcal{M}_y)$ does not necessarily create a connected feature diagram. In fact, this is the expected result, since in most cases the combined feature models do not share features ($\mathcal{F}_x \cap \mathcal{F}_y = \emptyset$). Thus, we need to describe the constraints between the two input feature models in a separate feature model $\mathcal{M}_C$ which can represent parent-child relationships that connects both feature models.

Let us consider the details of the aggregation regarding $\mathcal{M}_{DBMS}$ and $\mathcal{M}_{Index}$. In Figure 3, we aim to instantiate feature model $\mathcal{M}_{Index}$ in feature model $\mathcal{M}_{DBMS}$ below feature *Access* as an optional feature. Therefore, we have to transform feature model $\mathcal{M}_{Index}$ using the function R. Then, it is possible to combine both feature models using the function $\bullet$. We depict the result of this combination in Figure 3. Because of the desired parent-child dependency (i.e., *Index* ⇒ *Access*), we need to create a feature model $\mathcal{M}_C$ with a set of products that represents this dependency. Using the knowledge of the parent-child dependency, we create a feature model $\mathcal{M}_C$ with three products $\mathcal{P}_C = \{\emptyset, \{Access\}, \{Access, Index\}\}$. Additionally, we add two other constraints to further restrict the resulting products. Using the constraints

- *Index* ⇒ Paged
- *Multimedia* ⇒ *Knn*,

we create an extended version of feature model $\mathcal{M}_C$, which we depict in Figure 3 (in the left bottom corner). Now, we can use function $\bullet$ to eliminate all products of $\mathcal{M}_{DBMS} \bullet$

[1] If both feature sets are disjunct it is not possible to represent the result in an ordinary feature diagram because there is no connection between these models.

[2] Note that, if both feature set are disjunct, the condition $p \cap \mathcal{F}_y = q \cap \mathcal{F}_x = \emptyset$ is always true and, thus, the function works like a kind of "cross product" and creates all combinations of products.

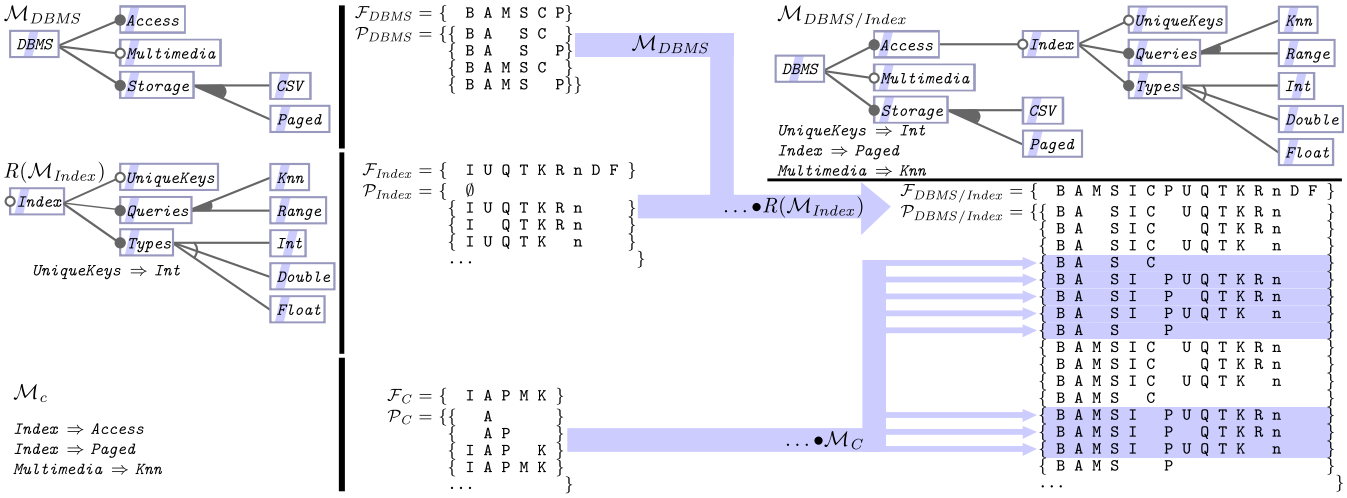

Figure 3: Aggregation of the feature models $\mathcal{M}_{DBMS}$ and $\mathcal{M}_{Index}$ using the feature model $\mathcal{M}_C$, which describes dependencies between both aggregated feature models.

$R(\mathcal{M}_{Index})$ that do not comply with the parent-child relationship and the other dependencies given in feature model $\mathcal{M}_C$. The result is a new feature model $\mathcal{M}_{DBMS/Index}$, which is also depicted in Figure 3.

4. PROBLEM STATEMENT

In the last sections, we gave an overview about automated analyses of feature models and the composition of feature models using the aggregation function. Both research domains present a set of approaches but their combination is rarely analyzed. In previous work, we and others proposed strategies in which feature models are combined to one single feature model so that we can reuse existing analyses [2, 24]. However, the necessity to combine feature models for automated analyses leads to several disadvantages and problems. In the following, we give some examples to illustrate them.

Scalability. Decomposition is one possibility to handle large feature models. As described before, we need to combine the separated feature models to analyze them. The result is again a large feature model that can be difficult to analyze. This depends not only on the analyses themselves but also on the strategy to apply them. For instance, we can use different solvers to apply a specific analysis, such as binary decision diagrams (BDDs) or satisfiability solvers. Let us take a look at the analysis *number of products*. If we use BDDs to determine the number of products, we typically have a scalability problem regarding the memory consumption. By contrast, if we use a satisfiability solver for the same analysis, we get a scalability problem regarding the time that is needed to determine the number.

Unused Feature Model Details. If we use feature-model composition to reuse existing functionality of one product line in another one (e.g., feature model $\mathcal{M}_{Index}$ in feature model $\mathcal{M}_{DBMS}$) it is possible that only some functionality is needed instead of the complete product line. This means, it is possible that features exist in the composed feature model that are not of the developer's interest (c.f. information hiding [14]). If we use such a composed feature model as input, the analyses are more complicated than needed, which, again, contributes to the problem of scalability.

Reusability of Analysis Results. Assume a scenario of our running example $\mathcal{M}_{Index}$ and $\mathcal{M}_{DBMS}$, in which both feature models are designed by completely independent development groups. If changes occur in the reused feature model $\mathcal{M}_{Index}$, the development group of feature model $\mathcal{M}_{DBMS}$ has to react on these changes to ensure a further usage. This also means that the new version of $\mathcal{M}_{Index}$ has to be recomposed with the feature model $\mathcal{M}_{DBMS}$ to apply the different analyses. With current techniques, it is not possible to reuse existing analyses of the first feature-model composition.

To conclude, it is necessary to find a way that allows us to modularize feature models such that compositional analyses are possible. A suitable modularization mechanism of feature models reduces the described problems above and facilitates the reuse of analysis results. To support compositional analyses, we introduce feature-model interfaces and prove a set of dependencies of analysis results based on feature-model interfaces to other feature models.

5. FEATURE-MODEL INTERFACES

In this section, we introduce feature-model interfaces - a mean to compositional analysis of feature models. Afterwards, we present a function that allows us to create feature-model interfaces based on existing feature models. We present and prove some algebraic properties regarding feature-model interfaces that we need for our proofs for compositional analyses. We define a feature-model interface as follows:

DEFINITION 3. *A feature model $\mathcal{M}_{Int} = (\mathcal{F}_{Int}, \mathcal{P}_{Int})$ is an interface of feature model $\mathcal{M}_x = (\mathcal{F}_x, \mathcal{P}_x)$ denoted as $\mathcal{M}_{Int} \preceq \mathcal{M}_x$, if and only if*

$$\mathcal{F}_{Int} \subseteq \mathcal{F}_x \text{ and}$$
$$\mathcal{P}_{Int} = \{p \cap \mathcal{F}_{Int} \mid p \in \mathcal{P}_x\}.$$

The definition of feature-model interfaces is based on the definition of feature models (see Section 2) because it is a fea-

ture model itself, which is related to another feature model $\mathcal{M}_x$. In detail, the feature-model interface $\mathcal{M}_{Int}$ has a possibly reduced set of features compared to feature model $\mathcal{M}_x$ and each product of $\mathcal{M}_{Int}$ is similar to a product of $\mathcal{M}_x$ but with a set of features compatible to $\mathcal{F}_{Int}$. Therefore, we can conclude that for each product in a feature model $\mathcal{M}_x$ a product in the feature-model interface $\mathcal{M}_{Int}$ exists and vice-versa.

Corollary 1.

$$\forall\, q \in \mathcal{P}_x \exists p \in \mathcal{P}_{Int} : p = q \cap \mathcal{F}_{Int}$$
$$\forall\, p \in \mathcal{P}_{Int} \exists q \in \mathcal{P}_x : p = q \cap \mathcal{F}_{Int}$$

Furthermore, we can conclude that for one feature model $\mathcal{M}_x$ and a set of features $\mathcal{F}_{Int}$ there exists exactly one feature-model interface.

5.1 Interface Generation

For our further investigation and proofs regarding automated analyses, we define a function S that allows us to generate a feature-model interface based on an existing feature model.

Definition 4. *Let $\mathcal{M}_x = (\mathcal{F}_x, \mathcal{P}_x)$ be a feature model and $\mathcal{F}_R$ a set of features, we define a function S that takes $\mathcal{M}_x$ and $\mathcal{F}_R$ as input and returns a feature model $\mathcal{M}_{Int}$ with $\mathcal{M}_{Int} \preceq \mathcal{M}_x$ and $\mathcal{F}_{Int} = \mathcal{F}_x \setminus \mathcal{F}_R$.*

$$S : \mathcal{M} \times 2^{\mathcal{F}} \to \mathcal{M}$$
$$\mathcal{M}_{Int} = S(\mathcal{M}_x, \mathcal{F}_R) = (\mathcal{F}_x \setminus \mathcal{F}_R, \{p \setminus \mathcal{F}_R \mid p \in \mathcal{P}_x\})$$

Function S takes as input a feature model $\mathcal{M}_x$ and a set of features $\mathcal{F}_R$ that are not of interest for a specific target domain and creates a new feature model $\mathcal{M}_{Int}$ as an interface of $\mathcal{M}_x$. In detail, the function S subtracts feature set $\mathcal{F}_R$ from $\mathcal{F}_x$ (i.e., feature set of feature model $\mathcal{M}_x$) and from all products in $\mathcal{P}_x$ and, by this, creates the one feature-model interface corresponding to $\mathcal{M}_x$ and the set of feature $\mathcal{F}_{Int}$ which is defined by $\mathcal{F}_x \setminus \mathcal{F}_R$.

Example. Let us consider our running example of the feature models $\mathcal{M}_{DBMS}$ and $\mathcal{M}_{Index}$. Again, the developers want to reuse an existing index structure for the enhancement of a database management system. However, some features are not of the developers' interest and, thus, they plan to reuse only parts of this product line. Therefore, they can use our feature-model interface (i.e., applying the function S) to reduce the set of features of the feature model $\mathcal{M}_{Index}$. In Figure 4, we illustrate the application of function S with feature model $\mathcal{M}_{Index}$ and a set of features $\mathcal{F}_R = \{Range, UniqueKeys, Float\}$ as input.[3] The result is a new feature model $\mathcal{M}_{Int}$ with a reduced set of features and products that is tailored to the developers' needs.

5.2 Algebraic Properties of Interfaces

Next, we take a look at certain properties of the function S that we need for our proofs of compositional analyses. In detail, we investigate the right identity for certain feature sets and distributivity of function S with the functions $\bullet$ and R.

[3]We assume the set $\mathcal{F}_R$ to be given. In practice, it depends on the specific reuse scenario and should be defined in cooperation with domain experts.

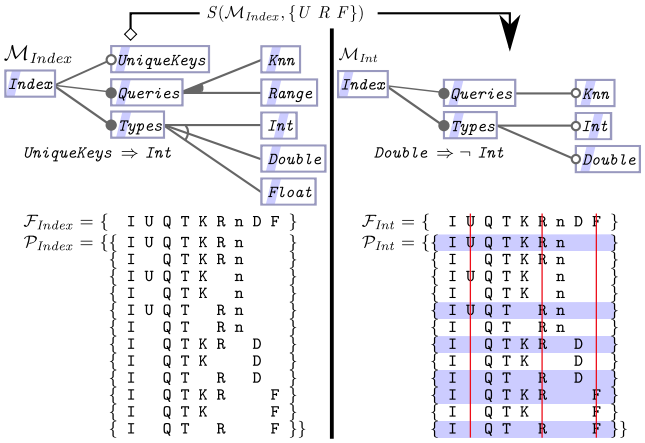

Figure 4: Application of function S with feature model $\mathcal{M}_{Index}$ and $\mathcal{F}_R$ as input. The highlighted products are part of the resulting feature-model interface $\mathcal{M}_{Int}$.

Right Identity. First, we prove that $\mathcal{F}_R$ is a right identity element to S if $\mathcal{F}_x$ does not contain any feature from $\mathcal{F}_R$. Therefore, we prove that function S has no effect on a feature model that does not contain a feature of the feature set $\mathcal{F}_R$.

Lemma 1. *Let $\mathcal{M}_x = (\mathcal{F}_x, \mathcal{P}_x)$ be a feature model and $\mathcal{F}_R$ a set of features with $\mathcal{F}_x \cap \mathcal{F}_R = \emptyset$, then*

$$S(\mathcal{M}_x, \mathcal{F}_R) = \mathcal{M}_x.$$

Proof. As the intersection of $\mathcal{F}_x$ and $\mathcal{F}_R$ is the empty set, there will be no feature that is removed from the set of features $\mathcal{F}_x$. The result is the identical feature set $\mathcal{F}_x$. Similarly, the intersection between each product and the set of features $\mathcal{F}_R$ is also empty and, thus, each product will be the same as before.

$$S((\mathcal{F}_x, \mathcal{P}_x), \mathcal{F}_R) = ((\mathcal{F}_x \setminus \mathcal{F}_R), \{p \setminus \mathcal{F}_R \mid p \in \mathcal{P}_x\}) \quad (4.1)$$
$$= (\mathcal{F}_x, \mathcal{P}_x) \quad (4.2)$$
$$= \mathcal{M}_x \quad (4.3)$$

□

Distributivity of $\bullet$ and S. Next, we prove that the order in which we apply the functions $\bullet$ and S is not relevant for the result.

Lemma 2. *Let $\mathcal{M}_x = (\mathcal{F}_x, \mathcal{P}_x)$, $\mathcal{M}_y = (\mathcal{F}_y, \mathcal{P}_y)$ be feature models and $\mathcal{F}_R$ a set of features*

$$S(\mathcal{M}_x \bullet \mathcal{M}_y, \mathcal{F}_R) = S(\mathcal{M}_x, \mathcal{F}_R) \bullet S(\mathcal{M}_y, \mathcal{F}_R).$$

Proof. In general, we separate the application of the function S on each part of the composed feature model so that we can apply function $\bullet$ later on. We start with the set of composed features and transform it accordingly.

$$S((\mathcal{F}_x, \mathcal{P}_x) \bullet (\mathcal{F}_y, \mathcal{P}_y), \mathcal{F}_R) \quad (5.1)$$
$$= (\mathcal{F}_z, \mathcal{P}_z) \quad (5.2)$$
$$= ((\mathcal{F}_x \cup \mathcal{F}_y) \setminus \mathcal{F}_R, \mathcal{P}_z) \quad (5.3)$$
$$= ((\mathcal{F}_x \setminus \mathcal{F}_R) \cup (\mathcal{F}_y \setminus \mathcal{F}_R), \mathcal{P}_z) \quad (5.4)$$

Next, we transform the definition of the product sets in a way that the feature sets r and s represent the transformation of function S which are used as input for function $\bullet$. Thus, r and s are in accordance to the Definition 3 and Eq. (5.8) is the application of Definition 2.

$$= (\mathcal{F}_z, \{(p \cup q) \setminus \mathcal{F}_R \mid p \in \mathcal{P}_x, q \in \mathcal{P}_y, p \cap \mathcal{F}_y = q \cap \mathcal{F}_x\}) \quad (5.5)$$

$$= (\mathcal{F}_z, \{(p \setminus \mathcal{F}_R) \cup (q \setminus \mathcal{F}_R) \mid p \in \mathcal{P}_x, q \in \mathcal{P}_y, p \cap \mathcal{F}_y = q \cap \mathcal{F}_x\}) \quad (5.6)$$

$$= (\mathcal{F}_z, \{r \cup s \mid r \in \{p \setminus \mathcal{F}_R \mid p \in \mathcal{P}_x\}, s \in \{q \setminus \mathcal{F}_R \mid q \in \mathcal{P}_y\}, r \cap \mathcal{F}_y = s \cap \mathcal{F}_x\}) \quad (5.7)$$

$$= S(\mathcal{M}_x, \mathcal{F}_R) \bullet S(\mathcal{M}_y, \mathcal{F}_R) \quad (5.8)$$

□

Distributivity of R and S. Finally, we prove that the order in which we apply the functions R and S is not relevant.

LEMMA 3. *Let $\mathcal{M}_x = (\mathcal{F}_x, \mathcal{P}_x)$ be a feature model and $\mathcal{F}_R$ a set of features, then*

$$S(R(\mathcal{M}_x), \mathcal{F}_R) = R(S(\mathcal{M}_x, \mathcal{F}_R)).$$

PROOF. Function R is used to add the empty set to the set of products of a given feature model. To prove the interaction, it is necessary to extract this empty set from the input feature model that is used for function S.

$$S(R(\mathcal{M}_x), \mathcal{F}_R)$$

$$= (\mathcal{F}_x \setminus \mathcal{F}_R, \{p \setminus \mathcal{F}_R \mid p \in (\mathcal{P}_x \cup \{\emptyset\})\}) \quad (6.1)$$

$$= (\mathcal{F}_x \setminus \mathcal{F}_R, \{p \setminus \mathcal{F}_R \mid p \in \mathcal{P}_x\} \cup \{\emptyset\}\}) \quad (6.2)$$

$$= R((\mathcal{F}_x \setminus \mathcal{F}_R, \{p \setminus \mathcal{F}_R \mid p \in \mathcal{P}_x\})) \quad (6.3)$$

$$= R(S(\mathcal{M}_x, \mathcal{F}_R)) \quad (6.4)$$

□

6. COMPOSITIONAL FEATURE-MODEL ANALYSES

In this section, we clarify how to support compositional analyses of feature models using feature-model interfaces. First, we present the general idea and illustrate how to combine aggregation (i.e., function $\circ$) and our concept of feature-model interfaces (i.e., function S). Second, we investigate whether it is possible to use the presented analysis operations of Section 2 in combination with feature-model interfaces for compositional analyses. In particular, we prove that the application of the analyses in combination with feature-model interface presents the intended results.

6.1 General Concept

In Section 3, we introduced the aggregation mechanism that we can use to instantiate one feature model within another. Afterwards, in Section 4, we presented an overview of existing problems regarding automated analysis based on combined feature models. Using a combination of a feature-model interface and our aggregation function, we show that it is possible to achieve compositional analyses for feature models.

To introduce our general concept of compositional analyses, we assume that two feature models $\mathcal{M}_x$ and $\mathcal{M}_y$ are composed to $\mathcal{M}_{x/y} = \mathcal{M}_x \circ_{\mathcal{M}_C} \mathcal{M}_y$. Typically, not all features of feature model $\mathcal{M}_y$ are of interest for the composition with feature model $\mathcal{M}_x$. Given the knowledge about those features, it is possible to create a feature-model interface $\mathcal{M}_{Int}$ based on $\mathcal{M}_y$ with all features of interest ($\mathcal{M}_{Int} \preceq \mathcal{M}_y$). Since feature model $\mathcal{M}_{Int}$ consists of all "important" features, the question arise whether it is possible to use feature model $\mathcal{M}_{Int}$ instead of feature model $\mathcal{M}_y$ for the feature-model composition with $\mathcal{M}_x$ (i.e., $\mathcal{M}_x \circ_{\mathcal{M}_C} \mathcal{M}_{Int} = \mathcal{M}_{x/Int}$). In detail, our goal is to prove specific dependencies between analysis results for automated analyses based on feature model $\mathcal{M}_{x/Int}$ and feature model $\mathcal{M}_{x/y}$. One dependency could be that we achieve exactly the same results for an analysis based on a feature-model interface $\mathcal{M}_{x/Int}$ compared to the same analysis based on feature model $\mathcal{M}_{x/y}$ with $\mathcal{M}_{Int} \preceq \mathcal{M}_y$.

For our proofs regarding the analysis-result dependencies of feature model $\mathcal{M}_{x/Int}$ and $\mathcal{M}_{x/y}$, we investigate and prove the dependencies between the analysis results of feature-model interface $\mathcal{M}_{Int}$ and feature model $\mathcal{M}_y$ ($\mathcal{M}_{Int} \preceq \mathcal{M}_y$). For this, we prove that a feature-model composition based on a feature-model interface $\mathcal{M}_{Int}$ is also an interface regarding a composition based on $\mathcal{M}_y$ (i.e., $\mathcal{M}_{x/Int} \preceq \mathcal{M}_{x/y}$). Based on that knowledge, we know the dependencies of the composed feature models $\mathcal{M}_{x/Int}$ and $\mathcal{M}_{x/y}$, because feature model $\mathcal{M}_{x/Int}$ can be considered as an ordinary feature-model interface of $\mathcal{M}_{x/y}$.

For each analysis that we discussed in Section 2, we identify a specific relation between the analysis results regarding the feature model $\mathcal{M}_y$ and its feature-model interface $\mathcal{M}_{Int}$ (e.g., $\mathcal{M}_{Int}$ is a void feature model if and only if $\mathcal{M}_y$ is void). The particular dependency of the analysis results regarding the feature models $\mathcal{M}_y$ and $\mathcal{M}_{Int}$ depends on the given analysis.

6.2 Feature-Model Interfaces in Compositions

Before we can start with the investigation of each analysis regarding the support of compositionality, we prove that a composed feature model $\mathcal{M}_{x/Int}$ using a feature-model interface $\mathcal{M}_{Int}$ ($\mathcal{M}_{Int} \preceq \mathcal{M}_y$) is also a feature-model interface of the $\mathcal{M}_{x/y}$ ($\mathcal{M}_{x/Int} \preceq \mathcal{M}_{x/y}$). In other words, we prove that the function S applied on a composed feature model $\mathcal{M}_{x/y}$ is identical to the application of function S on feature model $\mathcal{M}_y$ and a subsequent aggregation.

LEMMA 4. *Let $\mathcal{M}_{x/y} = \mathcal{M}_x \circ_{\mathcal{M}_C} \mathcal{M}_y$, $\mathcal{M}_{x/Int} = \mathcal{M}_x \circ_{\mathcal{M}_C} \mathcal{M}_{Int}$ be composed feature models based on the feature models $\mathcal{M}_x = (\mathcal{F}_x, \mathcal{P}_x)$, $\mathcal{M}_y = (\mathcal{F}_y, \mathcal{P}_y)$, $\mathcal{M}_C = (\mathcal{F}_C, \mathcal{P}_C)$, $\mathcal{M}_{Int} = S(\mathcal{M}_y, \mathcal{F}_R)$ with $\mathcal{F}_R \cap \mathcal{F}_x = \mathcal{F}_R \cap \mathcal{F}_C = \emptyset$, then:*

$$\mathcal{M}_{x/Int} \preceq \mathcal{M}_{x/y}$$

PROOF. Given the algebraic properties of the function S and the definition of our aggregation function $\circ_{\mathcal{M}_C}$, the following relations hold:

$$\mathcal{M}_{x/y} \succeq S(\mathcal{M}_{x/y}, \mathcal{F}_R) \quad (7.1)$$

$$= S(\mathcal{M}_x \circ_{\mathcal{M}_C} \mathcal{M}_y, \mathcal{F}_R) \quad (7.2)$$

$$\overset{(Eq.\ (1.1))}{=} S((\mathcal{M}_x \bullet R(\mathcal{M}_y)) \bullet \mathcal{M}_C, \mathcal{F}_R) \quad (7.3)$$

$$\overset{(Lemma\ 2)}{=} (S(\mathcal{M}_x, \mathcal{F}_R) \bullet S(R(\mathcal{M}_y), \mathcal{F}_R)) \bullet S(\mathcal{M}_C, \mathcal{F}_R) \quad (7.4)$$

$$\overset{(Lemma\ 1)}{=} (\mathcal{M}_x \bullet S(R(\mathcal{M}_y), \mathcal{F}_R)) \bullet \mathcal{M}_C \quad (7.5)$$

$$\overset{(Lemma\ 3)}{=} (\mathcal{M}_x \bullet R(S(\mathcal{M}_y, \mathcal{F}_R))) \bullet \mathcal{M}_C \quad (7.6)$$

$$\overset{(Definition\ 4)}{=} (\mathcal{M}_x \bullet R(\mathcal{M}_{Int}))) \bullet \mathcal{M}_C \tag{7.7}$$

$$\overset{(Eq.\ (1.1))}{=} \mathcal{M}_x \circ_{\mathcal{M}_C} \mathcal{M}_{Int} \tag{7.8}$$

$$= \mathcal{M}_{x/Int} \tag{7.9}$$

□

6.3 Relation of Analysis Results

In this section, we investigate each analysis that we formalized in Section 2. In detail, we investigate the dependencies of the analysis results for the analyses of *void feature model*, *core features*, *dead features*, and *valid partial configurations*. For each analysis, we start with an investigation of the analysis-result dependencies between feature model $\mathcal{M}_y$ and $\mathcal{M}_{Int}$ using the following premise:

PREMISE 1. *Let $\mathcal{M}_y = (\mathcal{F}_y, \mathcal{P}_y)$ and $\mathcal{M}_{Int} = S(\mathcal{M}_y, \mathcal{F}_R)$ $= (\mathcal{F}_{Int}, \mathcal{P}_{Int})$ be feature models. Thus, the feature model $\mathcal{M}_{Int}$ is a feature-model interface of $\mathcal{M}_y$ ($\mathcal{M}_{Int} \preceq \mathcal{M}_y$).*

Afterwards, we use the knowledge of Lemma 4 in which the same dependency holds for composed feature models. Therefore, we use a second premise:

PREMISE 2. *Let $\mathcal{M}_{x/y} = \mathcal{M}_x \circ_{\mathcal{M}_C} \mathcal{M}_y$, $\mathcal{M}_{x/Int} = \mathcal{M}_x \circ_{\mathcal{M}_C}$ $\mathcal{M}_{Int}$ be composed feature models based on the feature models $\mathcal{M}_x = (\mathcal{F}_x, \mathcal{P}_x)$, $\mathcal{M}_y = (\mathcal{F}_y, \mathcal{P}_y)$, $\mathcal{M}_C = (\mathcal{F}_C, \mathcal{P}_C)$, $\mathcal{M}_{Int} = S(\mathcal{M}_y, \mathcal{F}_R)$ with $\mathcal{F}_R \cap \mathcal{F}_x = \mathcal{F}_R \cap \mathcal{F}_C = \emptyset$.*

6.3.1 Void Feature Models

A feature-model interface $\mathcal{M}_{Int}$ with $\mathcal{M}_{Int} \preceq \mathcal{M}_y$ is a void feature model if and only if the feature model $\mathcal{M}_y$ is a void feature model.

THEOREM 1.

$$\mathcal{M}_y \in void \Leftrightarrow \mathcal{M}_{Int} \in void.$$

PROOF.
Because of Corollary 1, the following equations hold:

$$\mathcal{M}_y \in void \Leftrightarrow \mathcal{P}_y = \emptyset \tag{8.1}$$

$$\overset{(Corollary\ 1)}{\Leftrightarrow} \mathcal{P}_{Int} = \emptyset \tag{8.2}$$

$$\Leftrightarrow \mathcal{M}_{Int} \in void \tag{8.3}$$

□

Based on this knowledge, we take a closer look into the analysis of void feature models regarding composed feature models. Here, a feature model $\mathcal{M}_{x/Int}$ is a void feature model if $\mathcal{M}_{x/y}$ is a void feature model. Utilizing Premise 2, the following theorem holds:

THEOREM 2.

$$\mathcal{M}_{x/y} \in void \Leftrightarrow \mathcal{M}_{x/Int} \in void.$$

PROOF. From Lemma 4 and Theorem 1, we infer that the same analysis-result dependency is also valid for $\mathcal{M}_{x/Int}$ and $\mathcal{M}_{x/y}$. □

6.3.2 Core Features

With respect to Premise 1, a feature $f \in \mathcal{F}_{Int}$ is a core feature of feature model $\mathcal{M}_{Int}$ if and only if f is a core feature of feature model $\mathcal{M}_y$. Here, we use our function *core* and the resulting set of core features to prove the dependency.

THEOREM 3.

$$core(\mathcal{M}_y) \cap \mathcal{F}_{Int} = core(\mathcal{M}_{Int}).$$

PROOF. Based on Definition 3, the following equations hold:

$$core(\mathcal{M}_{Int}) = \bigcap_{p \in \mathcal{P}_{Int}} p \tag{10.1}$$

$$\overset{(Definition\ 3)}{=} \bigcap_{p \in \mathcal{P}_y} (p \cap \mathcal{F}_{Int}) \tag{10.2}$$

$$= (\bigcap_{p \in \mathcal{P}_y} p) \cap \mathcal{F}_{Int} \tag{10.3}$$

$$= core(\mathcal{M}_y) \cap \mathcal{F}_{Int} \tag{10.4}$$

□

Therefore, we can conclude that if a feature f is a core feature in feature model $\mathcal{M}_{Int}$ it is also a core feature in feature model $\mathcal{M}_y$. In addition, if we determine core features of feature model $\mathcal{M}_y$ that are also part of feature model $\mathcal{M}_{Int}$, it is also a core feature of feature model $\mathcal{M}_{Int}$.

$$f \in core(\mathcal{M}_{Int}) \Rightarrow f \in core(\mathcal{M}_y)$$
$$f \in core(\mathcal{M}_y) \cap \mathcal{F}_{Int} \Rightarrow f \in core(\mathcal{M}_{Int})$$

Using Theorem 3, we can take a closer look into composed feature models. With respect to Premise 2, a feature $f \in \mathcal{F}_{x/Int}$ is a core feature of $\mathcal{M}_{x/Int}$ if and only if f is a core feature in $\mathcal{M}_{x/y}$.

THEOREM 4.

$$core(\mathcal{M}_{x/y}) \cap \mathcal{F}_{x/Int} = core(\mathcal{M}_{x/Int}).$$

PROOF. From Lemma 4 and Theorem 3, we infer that the same analysis-result dependency is also valid for $\mathcal{M}_{x/Int}$ and $\mathcal{M}_{x/y}$. □

6.3.3 Dead Features

In compliance with Premise 1, a feature $f \in \mathcal{F}_{Int}$ is a dead feature of feature model $\mathcal{M}_{Int}$ if and only if f is a dead feature of feature model $\mathcal{M}_y$. Similar to Theorem 3, we use our formalization of function *dead* and the resulting set of dead features to prove this dependency.

THEOREM 5.

$$dead(\mathcal{M}_y) \cap \mathcal{F}_{Int} = dead(\mathcal{M}_{Int})$$

PROOF. Based on Definition 3, the following equation hold:

$$dead(\mathcal{M}_{Int}) = \mathcal{F}_{Int} \setminus \bigcup_{p \in \mathcal{P}_{Int}} p \tag{12.1}$$

$$\overset{(Definition\ 3)}{=} (\mathcal{F}_y \cap \mathcal{F}_{Int}) \setminus (\bigcup_{p \in \mathcal{P}_y} (p \cap \mathcal{F}_{Int})) \tag{12.2}$$

$$= (\mathcal{F}_y \cap \mathcal{F}_{Int}) \setminus ((\bigcup_{p \in \mathcal{P}_y} p) \cap \mathcal{F}_{Int}) \tag{12.3}$$

$$= (\mathcal{F}_y \setminus \bigcup_{p \in \mathcal{P}_y} p) \cap \mathcal{F}_{Int} \tag{12.4}$$

$$= dead(\mathcal{M}_y) \cap \mathcal{F}_{Int} \tag{12.5}$$

□

Therefore, a feature f that is a dead feature in the feature-model interface $\mathcal{M}_{Int}$ is also a dead feature in $\mathcal{M}_y$. Furthermore, if a feature f is a dead feature in feature model $\mathcal{M}_y$

and f is also part of the feature-model interface $\mathcal{M}_{Int}$, it is also a dead feature in feature-model interface $\mathcal{M}_{Int}$.

$$f \in dead(\mathcal{M}_{Int}) \Rightarrow f \in dead(\mathcal{M}_y)$$
$$f \in dead(\mathcal{M}_y) \cap \mathcal{F}_{Int} \Rightarrow f \in dead(\mathcal{M}_{Int})$$

Again, we take a look into the dependencies of analysis results regarding feature-model compositions. Using Premise 2, a feature $f \in \mathcal{F}_{x/Int}$ is a dead feature of feature model $\mathcal{M}_{x/Int}$ if and only if f is a dead feature of feature model $\mathcal{M}_{x/y}$.

THEOREM 6.

$$dead(\mathcal{M}_{x/y}) \cap \mathcal{F}_{x/Int} = dead(\mathcal{M}_{x/Int})$$

PROOF. From Lemma 4 and Theorem 5, we infer that the same analysis-result dependency is also valid for $\mathcal{M}_{x/Int}$ and $\mathcal{M}_{x/y}$. □

6.3.4 Valid Partial Configuration

Using Premise 1, a partial configuration $C = (\mathcal{F}_S, \mathcal{F}_D)$ with $\mathcal{F}_S \subseteq \mathcal{F}_{Int}$ and $\mathcal{F}_D \subseteq \mathcal{F}_{Int}$ is a valid partial configuration of feature model $\mathcal{M}_{Int}$ if and only if C is a valid partial configuration of feature model $\mathcal{M}_y$. We use our formalization of function *vConf* and the resulting set of configurations to prove this dependency.

THEOREM 7.

$$vConf(\mathcal{M}_{Int}) =$$
$$\{(\mathcal{F}_S \cap \mathcal{F}_{Int}, \mathcal{F}_D \cap \mathcal{F}_{Int}) \mid (\mathcal{F}_S, \mathcal{F}_D) \in vConf(\mathcal{M}_y)\}$$

PROOF. Based on Definition 3, the following equations hold:

$$\begin{aligned}
&vConf(\mathcal{M}_{Int}) \\
&\overset{(Definition)}{=} \{(\mathcal{F}_S, \mathcal{F}_D) \mid \exists p \in \mathcal{P}_{Int} : && \\
&\qquad \mathcal{F}_S \subseteq p \ \wedge \mathcal{F}_D \subseteq \mathcal{F}_{Int} \setminus p\} && (14.1)\\
&\overset{(Corollary\ 1)}{=} \{(\mathcal{F}_S, \mathcal{F}_D) \mid \exists q \in \mathcal{P}_y : && \\
&\qquad \mathcal{F}_S \subseteq q \cap \mathcal{F}_{Int} \ \wedge && \\
&\qquad \mathcal{F}_D \subseteq \mathcal{F}_{Int} \setminus (q \cap \mathcal{F}_{Int})\} && (14.2)\\
&\overset{(\mathcal{F}_{Int} \subseteq \mathcal{F}_y)}{=} \{(\mathcal{F}_S, \mathcal{F}_D) \mid \exists q \in \mathcal{P}_y : && \\
&\qquad \mathcal{F}_S \subseteq q \cap \mathcal{F}_{Int} \ \wedge && \\
&\qquad \mathcal{F}_D \subseteq (\mathcal{F}_y \cap \mathcal{F}_{Int}) \setminus (q \cap \mathcal{F}_{Int})\} && (14.3)\\
&= \{(\mathcal{F}_S, \mathcal{F}_D) \mid \exists q \in \mathcal{P}_y : && \\
&\qquad \mathcal{F}_S \subseteq q \cap \mathcal{F}_{Int} \ \wedge && \\
&\qquad \mathcal{F}_D \subseteq (\mathcal{F}_y \setminus q) \cap \mathcal{F}_{Int}\} && (14.4)\\
&= \{(\mathcal{F}_S \cap \mathcal{F}_{Int}, \mathcal{F}_D \cap \mathcal{F}_{Int}) \mid \exists q \in \mathcal{P}_y : && \\
&\qquad \mathcal{F}_S \subseteq q \ \wedge \mathcal{F}_D \subseteq \mathcal{F}_y \setminus q\} && (14.5)\\
&\overset{(Definition)}{=} \{(\mathcal{F}_S \cap \mathcal{F}_{Int}, \mathcal{F}_D \cap \mathcal{F}_{Int}) \mid && \\
&\qquad (\mathcal{F}_S, \mathcal{F}_D) \in vConf(\mathcal{M}_y)\} && (14.6)
\end{aligned}$$

□

As result, we know that each valid partial configuration of $\mathcal{M}_{Int}$ is also a valid partial configuration of $\mathcal{M}_y$. Furthermore, valid partial configurations of $\mathcal{M}_y$ are also valid partial configurations of $\mathcal{M}_{Int}$, if $\mathcal{F}_S$ and $\mathcal{F}_D$ are intersected with feature set $\mathcal{F}_{Int}$.

$$(\mathcal{F}_S, \mathcal{F}_D) \in vConf(\mathcal{M}_{Int}) \Rightarrow (\mathcal{F}_S, \mathcal{F}_D) \in vConf(\mathcal{M}_y)$$
$$(\mathcal{F}_S, \mathcal{F}_D) \in vConf(\mathcal{M}_y) \Rightarrow$$
$$(\mathcal{F}_S \cap \mathcal{F}_{Int}, \mathcal{F}_D \cap \mathcal{F}_{Int}) \in vConf(\mathcal{M}_{Int})$$

Based on Theorem 7 and Premise 2, we consider the relationship of analysis results of composed feature models. Therefore, a partial configuration with $\mathcal{F}_S \subseteq \mathcal{F}_{x/Int}$ and $\mathcal{F}_D \subseteq \mathcal{F}_{x/Int}$ is a valid partial configuration if and only if $(\mathcal{F}_S, \mathcal{F}_D)$ is a valid partial configuration for feature model $\mathcal{M}_{x/y}$.

THEOREM 8.

$$vConf(\mathcal{M}_{x/Int}) =$$
$$\{(\mathcal{F}_S \cap \mathcal{F}_{x/Int}, \mathcal{F}_D \cap \mathcal{F}_{x/Int}) \mid (\mathcal{F}_S, \mathcal{F}_D) \in vConf(\mathcal{M}_{x/y})\}$$

PROOF. From Lemma 4 and Theorem 7, we infer that the same analysis-result dependency is also valid for $\mathcal{M}_{x/Int}$ and $\mathcal{M}_{x/y}$. □

6.4 Discussion and Application

Using the investigated relations between the analysis results, we are able to address the problems given in Section 4. For instance, we can use feature-model interface $\mathcal{M}_{Int}$ instead of a feature model $\mathcal{M}_y$ with $\mathcal{M}_{Int} \preceq \mathcal{M}_y$ for a feature-model aggregation with feature model $\mathcal{M}_x$. This leads to several advantages. First, if a developer is only interested in a subset of features of an existing feature model, it is possible to create a feature-model interface that is tailored to the developer's needs. We can use the feature model for a feature-model aggregation. Second, if we use a feature model that is subject to evolution (e.g., developed by another company), it is possible that changes and updates can affect one of our existing feature-model compositions. In case that feature model $\mathcal{M}_y$ evolves to $\mathcal{M}'_y$, we only have to reanalyze our composed feature models that are based on $\mathcal{M}_y$ if the dependency to the feature-model interface $\mathcal{M}_{Int}$ is broken. By contrast, if the dependency between feature model $\mathcal{M}_y$ and feature-model interface $\mathcal{M}_{Int}$ is still confirm to $\mathcal{M}_{Int} \preceq \mathcal{M}'_y$, all existing analysis results of the composed feature models are still valid.

7. RELATED WORK

Here, we discuss different kinds of related work that exist in the domain of feature-model interfaces and feature-model composition, automated analysis of feature models, and multi software product lines.

Interface Definition

The interface definition that is used in this paper is based on the work of Acher et al. who introduce an operator to slice a feature model and, thus, to reduce the set of contained features [3]. In detail, the proposed *slice* operator uses a feature model as input to create a new feature model that only consists of a subset of features but with unchanged feature dependencies. However, Acher et al. do not investigate this operator for compositional analyses.

We already presented the main idea of feature-model interfaces and their feasibility to ease the process of automated analyses in our previous work [22]. In this overview, we present the feature-model interface as one part of an overall concept based on a set of interfaces for each development step of a software product line, such as the *syntactical* (vari-

able application programming interface) and the *behavioral product-line interface* (behavior of methods). By contrast to the previous representation in which we also discuss the combination of the feature-model interface with other interfaces (e.g., a syntactical product-line interface [22, 23]), we formalize and prove the dependencies of analysis results between a specific feature-model interface and the related feature model.

The most related concept to feature-model interface are feature-model views [11, 16, 21], which also consist of a subset of features based on a master feature model. In general, feature-model views can be used to ease the manageability of large scale feature models and allow domain experts to focus on relevant features during feature-model configuration. Thus, different views regarding one master feature model are combined to get a valid configuration based on the partial configurations of each view. By contrast, a feature-model interface can be an interface of a set of different feature models. For compositional analyses it is only important that the dependency between the interface and the feature model is confirm to our definition of a feature-model interface ($\preceq$).

Feature-Model Analysis

In general, there exists a bunch of research in the area of automated analysis of feature models. Here, we only want to give a small reference two a few papers that are able to give more insights to the topic.

The necessity for automated analyses of feature models was introduced together with the feature models themselves. Kang et al. already recognize that tool support is essential for the success of the feature-model concept to ensure their correctness [13]. The first tool support was based on Prolog using a fact base and composition rules [13]. By contrast, Batory describes the transformation of feature models into propositional formulas [6], which allows us to use satisfiability solvers for analyses. While the check for satisfiability of a proportional formula is an NP-complete problem, Mendonca et al. show that the satisfiability check in the domain of feature models scales well [17]. Nevertheless, it is an open question whether different kinds of product-line analyses (e.g, family-based analysis of product lines) can profit by the speed up of a specific analysis using feature-model interface.

Benavides et al. present a survey about existing analyses of feature models with several information regarding the analysis concept, tool support, and references to work on particular analyses [7]. However, the presented techniques do not support compositional analyses, which we introduce in this paper.

Feature-Model Composition

Hubaux et al. present an overview of separation of concerns in feature diagram languages [12]. This topic is closely related to the feature-model composition. Furthermore, Acher et al. present and compare a set of composition operators, such as a merge operator based on union and intersection of features [4]. The authors compare different possibilities for the implementation of these operators and compare their advantages and drawbacks [1].

Beside the consideration of composition operators in isolation, different modeling languages were proposed that allow us to combine feature models. Eichelberger and Schmid give an overview about textual-modeling languages that can be used for large scale variability modeling [9]. The authors identify six languages (among them, VELVET [19], TVL [8]) that support variability-model composition and compare the languages regarding their facility to support *composition*, *modularity*, and *evolution*. It is also possible to integrate the functionality of feature-model interface in these languages and, thus, to benefit from the support of compositional analyses. For instance, we prototypically integrated our feature-model interface in VELVET and we want to use this integration for evaluation purposes in future.

Multi Software Product Lines

In this paper, we only consider examples in which the composition of two feature models is needed. Nevertheless, it is possible to use the composition mechanisms in a broader way in several application scenarios. One scenario are multi software product lines, which are product lines consisting of multiple product lines [15, 18]. Holl et al. present a more detailed definition of a multi software product line - "a set of several self-contained but still interdependent product lines that together represent a large-scale or ultra-large-scale system" [10]. Furthermore, the authors summarize different capabilities of a multi software product line and present existing support of them [10]. In the domain of multi software product lines, we can use the composition of feature models based on feature-model interfaces to decouple the close dependency of involved feature models [22]. In addition, using the results of this paper, we know that the decoupling also affects the feature-model analyses.

8. CONCLUSION

Feature models are often used to describe the commonality and variability in product lines, but applying them in real-world scenarios scalability problems. Decomposition of feature models is used to reduce this drawback. However, this kind of divide and conquer strategy does not include feature-model analyses that are necessary to ensure the correctness of the feature models. In general, the automated analyses for feature-model compositions are based on composed feature models and must be reapplied if one part of the composed feature models changes.

In this paper, we present an analyses concept based on feature-model interfaces that allows us to support compositional analyses. In particular, feature-model interfaces only represent features that are of the developer's interest, which results in a reduced set of features and products that must be considered during feature-model composition. We prove that an analysis process based on feature-model interfaces let us conclude on the results of the original feature model with all original features. The concept of feature-model composition based on feature-model interfaces offers a better encapsulation of feature models and supports evolutionary changes.

ACKNOWLEDGMENTS This work is partially funded by BMBF grant 01IS14017B.

9. REFERENCES

[1] M. Acher, P. Collet, P. Lahire, and R. France. Comparing Approaches to Implement Feature Model Composition. In *Proceedings of the European Conference on Modelling Foundations and Applications (ECMFA)*, pages 3–19. Springer, 2010.

[2] M. Acher, P. Collet, P. Lahire, and R. B. France. A Domain-Specific Language for Managing Feature Models. In *Proceedings of the ACM Symposium Applied Computing (SAC)*, pages 1333–1340. ACM, 2011.

[3] M. Acher, P. Collet, P. Lahire, and R. B. France. Slicing Feature Models. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, pages 424–427. IEEE Computer Science, 2011.

[4] M. Acher, B. Combemale, P. Collet, O. Barais, P. Lahire, and R. B. France. Composing Your Compositions of Variability Models. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 352–369. Springer, 2013.

[5] S. Apel, D. Batory, C. Kästner, and G. Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer, 2013.

[6] D. Batory. Feature Models, Grammars, and Propositional Formulas. In *Proceedings of the International Software Product Line Conference (SPLC)*, pages 7–20. Springer, 2005.

[7] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Information Systems*, 35(6):615–708, 2010.

[8] A. Classen, Q. Boucher, and P. Heymans. A Text-based Approach to Feature Modelling: Syntax and Semantics of TVL. *Science of Computer Programming (SCP)*, 76(12):1130–1143, 2011.

[9] H. Eichelberger and K. Schmid. A Systematic Analysis of Textual Variability Modeling Languages. In *Proceedings of the International Software Product Line Conference (SPLC)*, pages 12–21. ACM, 2013.

[10] G. Holl, P. Grünbacher, and R. Rabiser. A Systematic Review and an Expert Survey on Capabilities Supporting Multi Product Lines. *J. Information and Software Technology (IST)*, 54(8):828–852, 2012.

[11] A. Hubaux, P. Heymans, P.-Y. Schobbens, and D. Deridder. Towards Multi-view Feature-Based Configuration. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 106–112. Springer, 2010.

[12] A. Hubaux, T. T. Tun, and P. Heymans. Separation of Concerns in Feature Diagram Languages: A Systematic Survey. *ACM Computing Surveys*, 45(4):51:1–51:23, 2013.

[13] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, 1990.

[14] C. Kästner, S. Apel, and K. Ostermann. The Road to Feature Modularity? In *Proceedings of the International SPLC Workshop Feature-Oriented Software Development (FOSD)*, pages 5:1–5:8. ACM, 2011.

[15] C. W. Krueger. New Methods in Software Product Line Development. In *Proceedings of the International Software Product Line Conference (SPLC)*, pages 95–102. IEEE Computer Science, 2006.

[16] M. Mannion, J. Savolainen, and T. Asikainen. Viewpoint-Oriented Variability Modeling. In *Proceedings of the Computer Software and Applications Conference (COMPSAC)*, pages 67–72. IEEE Computer Science, 2009.

[17] M. Mendonça, A. Wąsowski, and K. Czarnecki. SAT-Based Analysis of Feature Models is Easy. In *Proceedings of the International Software Product Line Conference (SPLC)*, pages 231–240. Software Engineering Institute, 2009.

[18] M. Rosenmüller and N. Siegmund. Automating the Configuration of Multi Software Product Lines. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 123–130. Universität Duisburg-Essen, 2010.

[19] M. Rosenmüller, N. Siegmund, T. Thüm, and G. Saake. Multi-Dimensional Variability Modeling. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 11–22. ACM, 2011.

[20] M. Rosenmüller, N. Siegmund, S. S. ur Rahman, and C. Kästner. Modeling Dependent Software Product Lines. In *Proceedings of the GPCE Workshop on Modularization, Composition and Generative Techniques for Product Line Engineering (McGPLE)*, pages 13–18. Department of Informatics and Mathematics, University of Passau, 2008.

[21] J. Schroeter, M. Lochau, and T. Winkelmann. Multi-Perspectives on Feature Models. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 252–268. Springer, 2012.

[22] R. Schröter, N. Siegmund, and T. Thüm. Towards Modular Analysis of Multi Product Lines. In *Proceedings of the International Software Product Line Conference co-located Workshops*, pages 96–99. ACM, 2013.

[23] R. Schröter, N. Siegmund, T. Thüm, and G. Saake. Feature-Context Interfaces: Tailored Programming Interfaces for Software Product Lines. In *Proceedings of the International Software Product Line Conference (SPLC)*, pages 102–111. ACM, 2014.

[24] R. Schröter, T. Thüm, N. Siegmund, and G. Saake. Automated Analysis of Dependent Feature Models. In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 9:1–9:5. ACM, 2013.

[25] R. Tartler, D. Lohmann, C. Dietrich, C. Egger, and J. Sincero. Configuration Coverage in the Analysis of Large-Scale System Software. *ACM SIGOPS Operating Systems Review*, 45(3):10–14, 2012.

[26] P. Trinidad and A. Ruiz-Cortés. Abductive Reasoning and Automated Analysis of Feature Models: How are They Connected? In *Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 145–153. Universität Duisburg-Essen, 2009.