Semantic Evolution Analysis of Feature Models

Imke Drave
Software Engineering, RWTH Aachen University
Aachen, Germany

Oliver Kautz
Software Engineering, RWTH Aachen University
Aachen, Germany

Judith Michael
Software Engineering, RWTH Aachen University
Aachen, Germany

Bernhard Rumpe
Software Engineering, RWTH Aachen University
Aachen, Germany

ABSTRACT

During the development process, feature models change continuously. Analyzing the semantic differences between consecutive feature model versions is important throughout the entire development process to detect unintended changes of the modeled product line. Previous work introduced a semantic differencing technique for feature models based on a closed-world assumption, which reveals the differences between two feature models when allowing products to only contain features used in the models. However, this does not reflect the stepwise refinement of feature models in early development stages. Therefore, we introduce an open-world semantics, an automatic method for semantic differencing of feature models with respect to the novel semantics, and formally relate the open- and closed-world semantics. We formally proof our results, including the relation between the different semantics as well as the soundness and completeness of the semantic differencing procedure. In conjunction with previous work, the results enable effective semantic feature model evolution analyses throughout the entire development process.

CCS CONCEPTS

• **Software and its engineering → Software system models**; **Formal methods**; **Semantics**; *Dynamic analysis*; *Design languages*; Software product lines.

KEYWORDS

feature modeling, model evolution analysis, semantic differences, open-world semantics

ACM Reference Format:
Imke Drave, Oliver Kautz, Judith Michael, and Bernhard Rumpe. 2019. Semantic Evolution Analysis of Feature Models. In *Proceedings of 23rd International Systems and Software Product Line Conference (SPLC'19).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Highly differentiated customer requirements challenge product selection, production, and delivery not only in software engineering, but also in, *e.g.,* production and logistics companies. Thus, configuring product families is nowadays standard in many business sectors, as it improves customer satisfaction and helps to achieve cost reduction. These configurations are not restricted to a certain domain or application and are applicable to any product with a high demand for variability, such as cars, bikes, laptops, and cameras. For these products, customers demand the ability to choose product features in a selection process provided by an online shop or an order process. In industry, software product line (SPL) engineering [14, 40] became a meaningful way to handle the development of a diversity of similar software applications [13, 17, 27, 44]. Feature modeling integrates the feature-oriented paradigm [4] for SPL development with model-driven engineering [20] by means of feature models (FMs) that represent all possible product configurations [22], similar to the 150% modeling approach [22].

In model-driven software engineering (MDSE), models are the primary development artifacts [20] and, thus, evolve over time. Refinement is an essential concept in MDSE. A model is a *refinement* of another model, if the semantics of the latter subsumes the semantics of the former [23]. Refinement steps in MDSE, *i.e.,* changing a model such that the new version is a refinement of the former version, are naturally performed in reaction to changing requirements and availability of additional information [26]. The idea is to start with an underspecified model encoding the available information and to iteratively refine the model once additional information becomes available, until ultimately obtaining a correct system implementation. Effective model evolution management is essential to detect bugs and explore design alternatives. Syntactic model evolution management is fairly established in modern MDSE development (*e.g.,* [3, 28–31, 45, 46]). Detecting semantic differences between two models enables to verify refinement with respect to the model's meaning and is subject to ongoing research (*e.g.,* [2, 12, 18, 19, 32, 33, 36–38]). Recent approaches combine syntactic with semantic differencing to identify the impact of syntactic changes on a model's semantics [21, 26, 34].

FMs describe features and the relations (mandatory, optional, alternative, exclusive, implies, excludes) between them. Propositional constraints restrict the sets of features of configurations possible in the FM [51, 52]. Existing approaches for semantic differencing of FMs [2] provide meaningful results under a *closed-world assumption* [41]. In the context of FMs, the closed-world assumption considers features not used in a FM to not exist. However, this assumption yields a quite restrictive semantics, especially in the context of model evolution analysis. To illustrate this, consider a restaurant offering several fixed dishes on a menu. In a closed-world, individual customer wishes, such as exchanging a supplement of a dish by a supplement of another one, would not be considered as an allowed configuration of the menu.

The *open-world assumption* considers FM semantics in a less restrictive way: Features that are not used in a FM are not considered to be non-existent, rather, it is assumed that their instantiation is not restricted. In the context of the restaurant example, in the open-world, exchanging one supplement by any other one, be it part of another dish or made up by the customer, would be considered a valid dish. In this case, it is more appropriate to stick to the open-world semantics, as it is less restrictive and thereby reflects the customer's intuition when choosing the dish. The fact that a dish is not explicitly mentioned on the menu does not mean that the restaurant's chef is not able to cook it. The open-world semantics as proposed in this paper should not replace the usual closed-world FM semantics but is to be seen as a complement to it, especially for performing analyses in early development stages.

In the following, Section 2 shows practical examples to motivate our idea and to illustrate the difference between closed- and open-world semantics. Section 3 introduces an abstract syntax of FMs, defines closed- and open-world semantics of FMs and analyses their relation. Section 4 introduces semantic FM differencing. Section 5 discusses our approach and its use in early development stages. Section 6 relates our approach to previous work concerning automated reasoning, semantic property analyses as well as model composition and synthesis. The last section concludes this paper and offers ideas for future research.

2 EXAMPLES

This section presents examples that illustrate potential use cases where open-world semantics are useful and that illustrate the difference between the closed- and open-world semantics.

A software developer was asked to create an online ordering service for a pizzeria. Therein, customers should be able to choose ingredients individually. During requirements engineering, the pizzeria owner and the web developer discuss all possible pizza configurations. The developer creates a FM that represents all available pizza configurations (*cf.* Figure 1, FM p_1): A pizza needs at least a base. The base can be either wheat or gluten-free. Optionally, customers can choose a sauce (tomato, hollandaise or nutella) and several toppings (banana, mozzarella, salami, broccoli). The implementation of all possible configurations in the online shop is based on the FM. Nevertheless, customers know the pizzeria quite well: They know, if other toppings are available at the restaurant, like ham, mushrooms, pineapple or rocket salad, it is possible to order them as well, via an additional phone call. In the real world, the FM modeling the pizzas available at the online store has an open-world semantics: Ingredient combinations, which are not explicitly forbidden by the FM, make up a pizza that can be ordered at the online shop.

The owner would like to make ham an "official" topping by offering it online as well, so the developer adds it as an optional feature of topping. Also, lately the pizzeria had been approached by several customers asking for sausage crust pizza, which is also added as an optional feature to the FM. Moreover, the cook refuses to put banana on any other sauce than nutella, which the developer implements by a requires-relation between banana and nutella. The FM $p_{1.2}$ in Figure 1 displays the adapted model.

However, the sales figures for sausage crust do not develop as expected. Thus, the owner instructs the developer to change it to cheesy crust instead (*cf.* $p_{1.3}$ in Figure 1). The tables in Figure 2 depict pizza configurations illustrating the difference between closed- and open-world semantics. The configuration c_1 = {pizza, base, wheat, topping, salami, ham} is contained in the open-world semantics of p_1 but not in its closed-world semantics. The configuration c_1 is, however, an element of the closed-world semantics of $p_{1.2}$ and, therefore, a diff witness for the closed-world semantic difference from $p_{1.2}$ to p_1. In the open-world, p_1 does not restrict including the ham-feature in valid configuration, therefore, ham-pizzas are available. Using a closed-world semantics, no ham-topped pizza are available in the p_1 based web-shop. Since customers were able to order ham pizza while p_1 implemented the available configurations, the closed-world semantics do not represent all available products. The Venn diagram depicted in Figure 3 illustrates that, indeed, $p_{1.2}$ and $p_{1.3}$ are both open-world refinements of p_1, while in the closed-world, all three have pairwise incomparable semantics. The former corresponds to the owner's and the developer's intuition of how the product range changes after adding the ham-topping and sausage crust-feature as well as the "requires" constraint between the nutella-sauce and the banana-topping. However, even in the open-world semantics, $p_{1.3}$ is no refinement of $p_{1.2}$ and vice versa. The configuration c_2 = {pizza, base, wheat, sauce, tomato, topping, banana} (*cf.* Figure 2) is a diff witness for both, the open-world and the closed-world semantic difference from p_1 to $p_{1.2}$. It is an element of the open- as well as the closed-world semantics of p_1, whereas it is no element of both, the open-world and the closed-world semantics of $p_{1.2}$. The "requires" constraint between the banana topping and the nutella sauce causes the witness. Using the closed-world semantics, the configuration c_3 = {pizza, base, wheat, cheesy crust}, is valid in FM $p_{1.3}$ but not valid in FM $p_{1.2}$. Using the open-world semantics, it is a valid configuration of both FMs. However, as Figure 3 shows, the two have incomparable open- and closed-world semantics.

The following sections define an abstract syntax for FMs and formalize both semantics and the difference between them.

3 FEATURE MODELS

This section introduces an abstract syntax for FMs and formally defines an open-world FM semantics. Further, it recaps the well-known closed-world FM semantics and relates the two semantics with each other.

3.1 An Abstract Syntax for Feature Models

Let U_F denote a countable (possibly infinite) universe of features. The elements of the set U_F represent feature names. The abstract syntax of FMs is defined as follows (similar to [51, 52]):

Definition 1 (Feature Model). *A feature model is a tuple* $FM = (F, E, r, child, I, EX)$ *where*

- $F \subseteq U_F$ *is a finite set of features,*

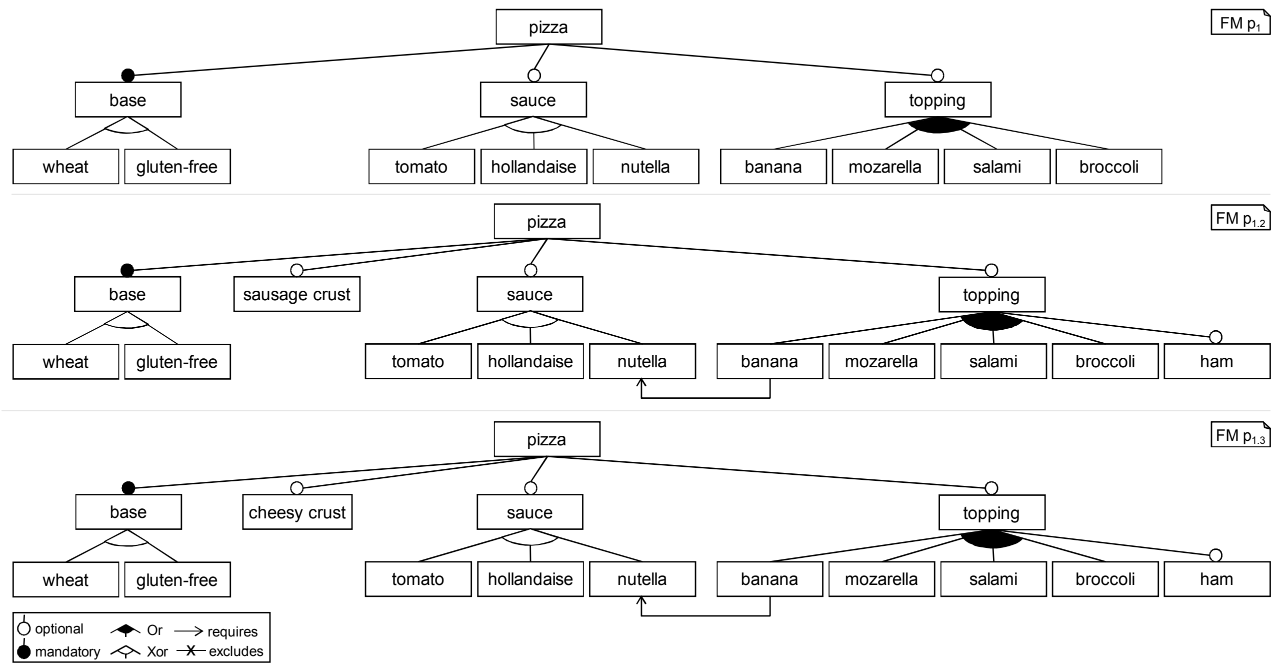


Figure 1: Three consecutive versions of the FM modeling pizzas available at the pizza store.

Closed World Semantics:

Configuration	$\in [\![p_1]\!]^{cw}$	$\in [\![p_{1.2}]\!]^{cw}$	$\in [\![p_1]\!]^{cw} \setminus [\![p_{1.2}]\!]^{cw}$	$\in [\![p_{1.2}]\!]^{cw} \setminus [\![p_1]\!]^{cw}$
C_1 = {pizza, base, wheat, topping, salami, ham}	✗	✓	✗	✓
C_2 = {pizza, base, wheat, sauce, tomato, topping, banana }	✓	✗	✓	✗

Open World Semantics:

Configuration	$\in [\![p_1]\!]^{ow}$	$\in [\![p_{1.2}]\!]^{ow}$	$\in [\![p_1]\!]^{ow} \setminus [\![p_{1.2}]\!]^{ow}$	$\in [\![p_{1.2}]\!]^{ow} \setminus [\![p_1]\!]^{ow}$
C_1 = {pizza, base, wheat, topping, ham}	✓	✓	✗	✗
C_2 = {pizza, base, wheat, sauce, tomato, topping, banana }	✓	✗	✓	✗

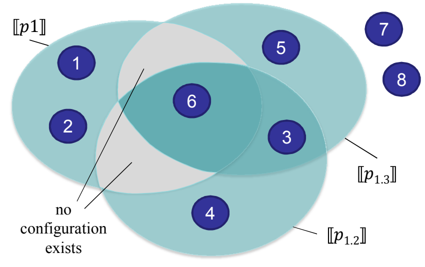
Figure 2: Configurations in the closed- and open-world semantics of the FMs depicted in Figure 1.

- *(F, E, r) is a directed rooted tree with nodes F, root $r \in F$, and edges $E \subseteq F \times F$,*
- *r is the root feature,*
- *$child : F \longrightarrow \wp(\wp(F))$ maps each feature $f \in F$ to its possible sets of children $child(f) \subseteq \wp(\{c \in F \mid (f, c) \in E\})$.*
- *$I \subseteq F \times F$ is a set of implies constraints,*
- *$EX \subseteq F \times F$ is a set of excludes constraints.*

For each $f \in F \setminus \{r\}$, we denote by $p_{FM}(f) \in F$ the parent of f in the feature model FM, i.e., for all features $f \in F \setminus \{r\}$, we define $p_{FM}(f) = p$ iff $(p, f) \in E$. If FM is known from the context, we simply write $p(f)$ instead of $p_{FM}(f)$.

Usually, the feature trees modeled in graphical notation are called feature diagrams and the combination of a feature diagram with a propositional formula is called FM (*e.g.,* [2]). The function *child* in the abstract syntax definition is not required to map every feature to sets of features resembling the usual groups used in graphical notation. This enables to describe every feature diagram in the usual graphical representation and additional constraints, for example, modeled with propositional formulas. Figure 4 depicts an example FM that can be formally defined as $(F, E, \mathsf{A}, child, I, Ex)$ with

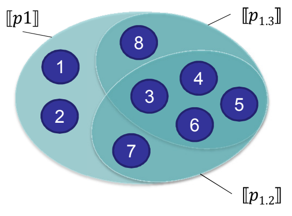
- the set of features $F = \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}, \mathsf{E}, \mathsf{F}, \mathsf{G}\}$,
- the root feature A,



a) closed world semantics

No.	Configuration
1	pizza, base, wheat, sauce, *hollandaise*, topping, *banana*
2	pizza, base, wheat, sauce, *tomato*, topping, *banana*
3	pizza, base, wheat, topping, *ham*
4	pizza, base, wheat, *sausage crust*
5	pizza, base, wheat, *cheesy crust*
6	pizza, base, wheat
7	*cheesy crust*
8	*sausage crust*

b) configuration examples



c) open world semantics

Figure 3: Closed- and open-world semantics for the pizza examples in Fig. 1

- the set of edges $E = \{(\mathsf{A},\mathsf{B}), (\mathsf{A},\mathsf{C}), (\mathsf{B},\mathsf{D}), (\mathsf{B},\mathsf{E}), (\mathsf{C},\mathsf{F}), (\mathsf{C},\mathsf{G})\}$,
- the function $child = \{\mathsf{A} \mapsto \{\{\mathsf{B}\},\{\mathsf{B},\mathsf{C}\}\}, \mathsf{B} \mapsto \{\{\mathsf{D}\},\{\mathsf{E}\}\}, \mathsf{C} \mapsto \{\{\mathsf{F}\},\{\mathsf{G}\},\{\mathsf{F},\mathsf{G}\}\}, \mathsf{D} \mapsto \emptyset, \mathsf{E} \mapsto \emptyset, \mathsf{F} \mapsto \emptyset, \mathsf{G} \mapsto \emptyset\}$,
- the set of implies constraints $I = \{(\mathsf{F},\mathsf{E})\}$, and
- the set of excludes constraints $Ex = \{(\mathsf{D},\mathsf{G})\}$.

3.2 Feature Model Semantics

The semantics of FMs are defined in terms of sets of configurations.

Definition 2 (Feature Configuration). *A (feature) configuration is a finite set of features $C \subseteq U_F$.*

The following defines when a configuration is considered to be possible in a FM with respect to the closed- and open-world semantics. We distinguish between configurations that are *products* of a FM and configurations that are *valid* in a FM. Intuitively, a configuration is valid in a FM, if it does not violate any constraint induced by the FM. A configuration is a product of a FM, if it does not violate any constraints induced by the FM and contains only features that are also contained in the FM. Consequently, every product of a FM is also valid in the FM. The valid configurations of a FM are the elements of its open-world semantics:

Definition 3 (Open-World Semantics). *A configuration $C \subseteq U_F$ is said to be valid in a FM $FM = (F, E, r, child, I, EX)$ iff*

(1) $\forall f \in C \cap F : f \neq r \Rightarrow p(f) \in C$,
(2) $\forall f \in C \cap F : \{g \in C \mid p(g) = f\} \in child(f)$,
(3) $\forall (f,g) \in I : f \in C \Rightarrow g \in C$,
(4) $\forall (f,g) \in EX : f \in C \Rightarrow g \notin C$.

The open-world semantics $[\![FM]\!]^{ow}$ of FM is defined as the set of all configurations that are valid in FM.

The first condition requires that if the configuration contains a non-root feature of the FM, then the configuration must also contain the feature's parent feature. Valid configurations are not required to contain the root feature. The second condition states that all maximal sets of features with a common parent are possible sets of child features of the parent features. The last two conditions state that requires and excludes constraints must be respected.

The closed-world semantics for FMs (*e.g.,* [2, 5, 8, 16, 55]) requires that all features of all configurations in the semantics of a FM must be contained in the feature set of the FM. Thus, the closed-world semantics are more constraining than the open-world semantics. The *products* of a FM are the configurations that are derived from the features contained in the FM and the constraints imposed by the FM. The closed-world semantics are formally defined as follows:

Definition 4 (Closed-World Semantics). *A configuration $C \subseteq U_F$ is called a product of a FM $FM = (F, E, r, child, I, EX)$ iff*

(1) $C \subseteq F$ and
(2) C is valid in FM.

The closed-world semantics $[\![FM]\!]^{cw}$ of FM is defined as the set of all products of FM.

The first condition in Definition 4 requires that all features of a FM's product must be elements of the FM. Thus, if a feature does not exist in a FM, then the feature is also assumed to be non-existent in any configuration of the FM's closed-world semantics. In contrast, using the open-world semantics, the feature is considered to be unconstrained. The difference between the closed-world semantics and the open-world semantics seems to be small. However, the choice of the semantics has strong implications in the context of model evolution analysis as discussed in Section 4.

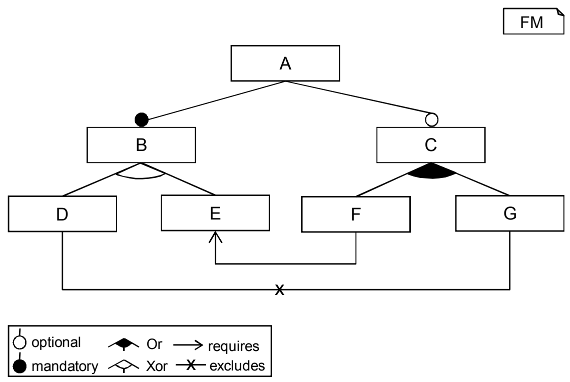
Each product of the FM depicted in Figure 4 may only contain features of the set $F = \{\mathsf{A},\mathsf{B},\mathsf{C},\mathsf{D},\mathsf{E},\mathsf{F},\mathsf{G}\}$. It must not contain other features. Configurations containing the feature H, for instance, are no valid products of the FM according to Definition 4. For example, the configuration $\{\mathsf{A},\mathsf{B},\mathsf{D}\}$ is a product of the FM. It satisfies all constraints induced by the FM and all features in the configuration are also used in the FM. In contrast, the configuration $\{\mathsf{A},\mathsf{B},\mathsf{E},\mathsf{H}\}$ is valid in the FM but no product of the FM: Although the configuration satisfies all constraints induced by the FM, it does not exclusively contain features that are used in the FM. The configuration $\{\mathsf{A},\mathsf{B},\mathsf{C},\mathsf{D},\mathsf{G}\}$ is neither valid in the FM nor a product of the FM, because it violates the FM's excludes constraint.

Considering the open-world semantics, features not used in a FM are unconstrained by the FM. Thus, removing features that are not used in an FM from a valid configuration of the FM yields a valid configuration of the FM.

Lemma 1. *Let $FM = (F, E, r, child, I, EX)$ be a feature model. Further, let $C \subseteq U_F$ and $C' \subseteq C$ be two configurations with $C \cap F = C' \cap F$. If $C \in [\![FM]\!]^{ow}$, then $C' \in [\![FM]\!]^{ow}$.*

Proof. Let C, C' and FM be given as above. Assume $C \in [\![FM]\!]^{ow}$. Then, by Definition 3, the following is satisfied:

(1) $\forall f \in C \cap F : f \neq r \Rightarrow p(f) \in C$,
(2) $\forall f \in C \cap F : \{g \in C \mid p(g) = f\} \in child(f)$,



F = {A, B, C, D,E, F, G} r = A I = {(F, E)} EX = {(G, D)}

Configuration	$\in [\![\mathbf{FM}]\!]^{cw}$	$\in [\![\mathbf{FM}]\!]^{ow}$	$\in [\![\mathbf{FM}]\!]^{cw} \cap [\![\mathbf{FM}]\!]^{ow}$
$C_1 = \{A, B, D\}$	✓	✓	✓
$C_2 = \{A, B, D, C, G\}$	✗	✗	✗
$C_3 = \{A, B, D, C, H\}$	✗	✓	✗

Figure 4: Feature model containing all quintessential modeling elements.

(3) $\forall (f, g) \in I : f \in C \Rightarrow g \in C$,
(4) $\forall (f, g) \in EX : f \in C \Rightarrow g \notin C$.

(a) As $C \cap F = C' \cap F$ and $p(f) \in F$ for all $f \in F$, (1) implies $\forall f \in C' \cap F : f \neq r \Rightarrow p(f) \in C$.

(b) As $C \cap F = C' \cap F$ and $p(f) \in F$ for all $f \in F$, we have $\{g \in C \mid p(g) = f\} = \{g \in C' \mid p(g) = f\}$. Using this and (2), we obtain that $\forall f \in C' \cap F : \{g \in C' \mid p(g) = f\} \in child(f)$.

(c) As $C \cap F = C' \cap F$ and $I \subseteq F \times F$, (3) implies $\forall (f, g) \in I : f \in C' \Rightarrow g \in C'$.

(d) As $C \cap F = C' \cap F$ and $EX \subseteq F \times F$, (4) implies $\forall (f, g) \in EX : f \in C' \Rightarrow g \notin C'$.

From (a)-(d) and Definition 3, we can conclude $C' \in [\![FM]\!]^{ow}$. □

Similarly, adding features that are not contained in a FM to a valid configuration of the FM yields a valid configuration.

Lemma 2. *Let $FM = (F, E, r, child, I, EX)$ be a feature model. Further, let $C \subseteq U_F$ and $N \subseteq U_F \setminus F$ be two configurations. If $C \in [\![FM]\!]^{ow}$, then $C \cup N \in [\![FM]\!]^{ow}$.*

Proof. Let FM, C, and N be given as above.

Assume $C \in [\![FM]\!]^{ow}$. As $N \cap F = \emptyset$, we have that $(C \cup N) \cap F = C \cap F$. As $\forall g \in F : p(g) \in F$ and since $N \cap F = \emptyset$, it holds that $\{g \in C \mid p(g) = f\} = \{g \in C \cup N \mid p(g) = f\}$ for all $f \in F$. With $C \in [\![FM]\!]^{ow}$ and using the above, we obtain

(a) $\forall f \in (C \cup N) \cap F : f \neq r \Rightarrow p(f) \in (C \cup N)$ and
(b) $\forall f \in (C \cup N) \cap F : \{g \in (C \cup N) \mid p(g) = f\} \in child(f)$.

As $N \cap F = \emptyset$, we have that $f \in C \cup N \Leftrightarrow f \in C$ for all $f \in F$. As further $C \in [\![FM]\!]^{ow}$, we obtain

(c) $\forall (f, g) \in I : f \in (C \cup N) \Rightarrow g \in (C \cup N)$ and
(d) $\forall (f, g) \in EX : f \in (C \cup N) \Rightarrow g \notin (C \cup N)$.

From (a)-(d) we can conclude that $C \cup N \in [\![FM]\!]^{ow}$. □

3.3 On the Relation between the Closed World Semantics and the Open World Semantics

The open-world semantics is more liberal than the closed-world semantics: Every product of a FM is also a valid configuration of the FM. The other direction does not hold.

Lemma 3. *$[\![FM]\!]^{cw} \subseteq [\![FM]\!]^{ow}$ for all feature models FM.*

Proof. Let FM be a FM and let $C \in [\![FM]\!]^{cw}$. By definition, C is valid in FM. Thus, $C \in [\![FM]\!]^{ow}$. □

Independent of the feature universe U_F, the closed-world semantics of every FM is always finite. If the universe of features U_F is infinite, then the open-world semantics of a FM is always infinite. Thus, enumerating all the products of a FM is always possible, whereas enumerating all of the FM's valid configurations is usually not possible. In contrast, if the universe of features U_F is finite, then the open-world semantics of every FM is also finite. In the special case where the feature universe U_F is finite and the FM uses all features of the universe, the open-world semantics and the closed-world semantics of the FM coincide.

Lemma 4. *For every feature model $FM = (F, E, r, child, I, EX)$, the following statements hold:*

(1) $[\![FM]\!]^{cw}$ is a finite set.
(2) If U_F is infinite, then $[\![FM]\!]^{ow}$ is infinite.
(3) If U_F is finite, then $[\![FM]\!]^{ow}$ is finite.
(4) If $U_F = F$, then $[\![FM]\!]^{cw} = [\![FM]\!]^{ow}$.

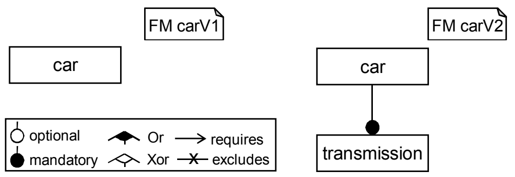
Proof. Let $FM = (F, E, r, child, I, EX)$ be a FM.

(1) By Definition 4, for every configuration $C \in [\![FM]\!]^{cw}$, it holds that $C \subseteq F$. Thus, $[\![FM]\!]^{cw} \subseteq \wp(F)$. As F is finite, $\wp(F)$ is finite, and thus $[\![FM]\!]^{cw}$ is finite.

(2) Assume U_F is infinite. The set of features F is by definition finite. Thus, the set $U_F \setminus F$ of features that are not used in the feature model FM is infinite. Using Definition 3, it is easy to verify that $\forall f \in U_F \setminus F : \{f\} \in [\![FM]\!]^{ow}$, *i.e.*, each configuration containing one feature not used in FM is valid in FM. As $U_F \setminus F$ is infinite, this implies $[\![FM]\!]^{ow}$ is infinite.

(3) Assume U_F is finite. Then, $\wp(U_F)$ is finite. As $[\![FM]\!]^{ow} \subseteq \wp(U_F)$, we can conclude that $[\![FM]\!]^{ow}$ is finite.

(4) Assume $U_F = F$. In particular, this implies that U_F is finite since F is finite by definition. By Lemma 3 we have that $[\![FM]\!]^{cw} \subseteq [\![FM]\!]^{ow}$. It remains to show that $[\![FM]\!]^{ow} \subseteq [\![FM]\!]^{cw}$. Let $C \in [\![FM]\!]^{ow}$ be a valid configuration of FM. As $U_F = F$ and $C \subseteq U_F$, we have that $C \subseteq F$. As C is valid in FM and $C \subseteq F$, we can conclude with Definition 4 that $C \in [\![FM]\!]^{cw}$. □



Configuration	$\in [\![carV1]\!]^{cw}$	$\in [\![carV1]\!]^{ow}$	$\in [\![carV2]\!]^{cw}$	$\in [\![carV2]\!]^{ow}$
$\{car\}$	✓	✓	✗	✗
$\{car, transmission\}$	✗	✓	✓	✓

Figure 5: Simple feature models possible in early development stages.

Further, the open-world semantics is antitone (order-reversing) with respect to the elements modeled in the FMs: The addition of information to a FM further restricts its set of valid configurations. If one FM contains at least all of the modeling elements that another FM contains, then the former FM's open-world semantics is a subset of the latter FM's open-world semantics. Especially in early development stages, this is a highly desired property as the addition of information (requirements) should only restrict the set of possible realizations, *i.e.,* valid configurations. This property does not necessarily hold for the closed-world semantics as illustrated in Figure 5. Every modeling element of the FM carV1 is also present in the FM carV2, but the closed-world semantics of carV2 is no subset of the closed-world semantics of carV1. On the other hand, the open-world semantics of carV2 is a subset of the open-world semantics of carV1. Hence, carV2 is an open-world but no closed-world refinement of carV1.

4 SEMANTIC DIFFERENCING OF FEATURE MODELS FOR EVOLUTION ANALYSES

Model evolution analysis is an important task to detect bugs and explore design alternatives. Syntactic differencing approaches [3] are not concerned with the semantics of modeling languages. They reveal the syntactic elements that have changed between two successor model versions. The syntactic difference from one FM to another FM reveals syntactic changes that yield the latter FM when they are applied to the former FM [3]. Thus, the result of a syntactic differencing approach for FMs is independent of the semantics under consideration. In contrast, semantic differencing [2, 26, 35] abstracts from the syntax of models. The semantic difference from one model to another model contains the elements in the semantics of the former model that are not elements in the semantics of the latter model. Thus, semantic differencing reveals the models' differences by elements of their meanings. The semantic difference from one FM to another FM is the set of elements of the one FM's semantics that are no members of the other FM's semantics:

Definition 5 (Semantic Difference). *Let FM_1 and FM_2 be two feature models.*

The closed-world semantic difference from FM_1 to FM_2 is defined as $\delta^{cw}(FM_1, FM_2) \overset{\text{def}}{=} [\![FM_1]\!]^{cw} \setminus [\![FM_2]\!]^{cw}$.

The open-world semantic difference from FM_1 to FM_2 is defined as $\delta^{ow}(FM_1, FM_2) \overset{\text{def}}{=} [\![FM_1]\!]^{ow} \setminus [\![FM_2]\!]^{ow}$.

Depending on the semantics under consideration, semantic differencing may yield different results. Previous work produced a semantic differencing operator for FMs using the closed-world semantics [2, 11] to decide whether $\delta^{cw}(FM_1, FM_2) = \emptyset$ for two FMs FM_1 and FM_2. This operator is especially useful when analyzing the semantic differences between two FMs in late development stages. By then, it is assumed that all possible features of the domain of interest are identified and contained in the FM modeling the product line. However, in early development stages, such as a specification phase in classical development processes, or when applying agile methods, requirements are subject to change. This also affects the set of possible features available in a product line during product line engineering, *i.e.,* the set of all features of the domain of interest is usually not known a priori. In this case, product line developers have an "open-world" view on the semantics of the product line under development - all features that are not explicitly constrained, are possible. In contrast, when using the closed-world semantics, all features that are not explicitly constrained, are not possible. For this reason, the existing operator for semantic differencing of FMs [2] using the closed-world semantics does not always yield intuitive results when used in early development stages.

For example, consider a car manufacturer starting with the very underspecified FM carV1 as depicted on the left hand side of Figure 5. This FM contains the single root feature car. Later in the development process, the manufacturer identifies that different car configurations support different types of transmission systems. The manufacturer thus changes the FM to carV2 as depicted on the right hand side of Figure 5. The manufacturer is still in the specification phase as the set of all possible car features is still to be identified. Therefore, as information is added to the model and no information is removed, the manufacturer would consider the FM carV2 to be a refinement of the FM carV1, *i.e.,* the manufacturer would expect that the semantics of carV2 is a subset of the semantics of carV1. This is the case when using the open-world semantics, *i.e.,* δ^{ow}(carV2, carV1) $= \emptyset$. In contrast, the FM carV2 is no refinement of its predecessor version carV1 when using the closed-world semantics, because {car, transmission} $\in \delta^{cw}$(carV2, carV1). On the other hand, in late development stages, when all possible features are known, the closed-world semantics is useful for detecting whether new products have been explicitly added or removed - which might be a circumstance that remains undetected when using the open-world semantics. Thus, depending on the development stage, both semantics provide meaningful results for semantic differencing.

In general, it is not possible to conclude properties of the open-world semantic difference between two FMs based on the closed-world semantic difference between the same FMs, and vice versa: If one FM is a refinement of another FM using the closed-world semantics, then the latter FM is not necessarily a refinement of the former FM using the open-world semantics, and vice versa. Figure 6 illustrates this: The FMs $FM1$ and $FM2$ have the same open-world semantics. However, neither of the two FMs is a refinement of the other FM using the closed-world semantics. Vice versa, the FM $FM2$ is a refinement of the FM $FM3$ using the closed-world semantics. However, under the open-world semantics the situation is reversed, *i.e.,* $FM3$ is a refinement of $FM2$ under the open-world semantics. However, if the two FMs under analysis share exactly the same features, then the two semantics are equivalent up to semantic differences, *i.e.,* the closed-world semantic difference from the one FM to the other FM is empty if, and only if, the open-world semantic difference from the FM to the other FM is empty.

Lemma 5. *Let* $FM_1 = (F_1, E_1, r_1, child_1, I_1, EX_1)$ *and* $FM_2 = (F_2, E_2, r_2, child_2, I_2, EX_2)$ *be two feature models with* $F_1 = F_2$. *Then,* $\delta^{cw}(FM_1, FM_2) = \emptyset$ *iff* $\delta^{ow}(FM_1, FM_2) = \emptyset$.

Proof. Let FM_1 and FM_2 be given as above.

"$\Rightarrow$": Assume $\delta^{cw}(FM_1, FM_2) = \emptyset$. Let $c \in [\![FM_1]\!]^{ow}$ be a configuration that is valid in FM_1. Using Lemma 1, we obtain that $c \cap F_1$ is also valid in FM_1. As $c \cap F_1$ is valid in FM_1 and $c \cap F_1 \subseteq F_1$, it follows that $c \cap F_1 \in [\![FM_1]\!]^{cw}$ is a product of FM_1. As by assumption $\delta^{cw}(FM_1, FM_2) = \emptyset$, we obtain that $c \cap F_1 \in [\![FM_2]\!]^{cw}$ is also a product of FM_2. Therefore, as $F_1 = F_2$, $c \cap F_2 = c \cap F_1 \in [\![FM_2]\!]^{ow}$ is valid in FM_2. Using Lemma 2, we obtain that $c = (c \cap F_2) \cup (c \setminus F_2) \in [\![FM_2]\!]^{ow}$ is also valid in FM_2. From the above, we can conclude that every valid configuration of FM_1 is also a valid configuration of FM_2 and, thus, $\delta^{ow}(FM_1, FM_2) = \emptyset$.

"$\Leftarrow$": Assume $\delta^{ow}(FM_1, FM_2) = \emptyset$. Let $c \in [\![FM_1]\!]^{cw}$ be a product of FM_1. Then, by definition $c \in [\![FM_1]\!]^{ow}$ is also valid in FM_1. As by assumption $\delta^{ow}(FM_1, FM_2) = \emptyset$, this implies $c \in [\![FM_2]\!]^{ow}$. Therefore, c is valid in FM_2. As further, $c \in [\![FM_1]\!]^{cw}$, it holds that $c \subseteq F_1 = F_2$. As $c \subseteq F_2$ and c is valid in FM_2, we can conclude that $c \in [\![FM_2]\!]^{cw}$. From the above, we can conclude that every product of FM_1 is also a product of FM_2 and, thus, $\delta^{cw}(FM_1, FM_2) = \emptyset$. □

The following introduces a semantic differencing operator for FMs using the open-world semantics. The operator is an automatic method for checking whether $\delta^{ow}(FM_1, FM_2) = \emptyset$ for any two FMs FM_1 and FM_2. It further yields a witness $w \in \delta^{ow}(FM_1, FM_2)$, if $\delta^{ow}(FM_1, FM_2) = \emptyset$ does not hold. The enabler for the open-world semantic differencing method is that it suffices to search a finite set of configurations for a configuration that is valid in the one FM and not valid in the other FM. More specifically, it suffices to search all possible configurations that solely contain features that exist in the input FMs:

Theorem 1. *Let* $FM_1 = (F_1, E_1, r_1, child_1, I_1, EX_1)$ *and* $FM_2 = (F_2, E, r_2, child_2, I_2, EX_2)$ *be two feature models. Then,* $[\![FM_1]\!]^{ow} \subseteq [\![FM_2]\!]^{ow}$ *iff* $([\![FM_1]\!]^{ow} \cap \wp(F_1 \cup F_2)) \subseteq ([\![FM_2]\!]^{ow} \cap \wp(F_1 \cup F_2))$.

Proof. Let FM_1 and FM_2 be given as above.

"$\Rightarrow$": Assume $[\![FM_1]\!]^{ow} \subseteq [\![FM_2]\!]^{ow}$ holds. This directly implies $([\![FM_1]\!]^{ow} \cap \wp(F_1 \cup F_2)) \subseteq ([\![FM_2]\!]^{ow} \cap \wp(F_1 \cup F_2))$.

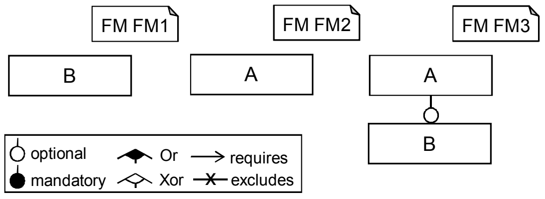
"$\Leftarrow$": Assume $([\![FM_1]\!]^{ow} \cap \wp(F_1 \cup F_2)) \subseteq ([\![FM_2]\!]^{ow} \cap \wp(F_1 \cup F_2))$ holds. Let $C \in [\![FM_1]\!]^{ow}$ be an arbitrary configuration that is valid in FM_1. We define $C' \overset{\text{def}}{=} C \cap (F_1 \cup F_2)$. As $C \in [\![FM_1]\!]^{ow}$ and $C' \cap F_1 = (C \cap (F_1 \cup F_2)) \cap F_1 = C \cap F_1$, using Lemma 1, it is guaranteed that $C' \in [\![FM_1]\!]^{ow}$. Therefore, as $C' \subseteq F_1 \cup F_2$, we have that $C' \in [\![FM_1]\!]^{ow} \cap \wp(F_1 \cup F_2)$. As by assumption $([\![FM_1]\!]^{ow} \cap \wp(F_1 \cup F_2)) \subseteq ([\![FM_2]\!]^{ow} \cap \wp(F_1 \cup F_2))$, we obtain that $C' \in [\![FM_2]\!]^{ow} \cap \wp(F_1 \cup F_2)$. We observe that $C \setminus C' = C \setminus (C \cap (F_1 \cup F_2)) = C \setminus (F_1 \cup F_2) \subseteq U_F \setminus (F_1 \cup F_2) \subseteq U_F \setminus F_2$. As also $C' \in [\![FM_2]\!]^{ow} \cap \wp(F_1 \cup F_2) \subseteq [\![FM_2]\!]^{ow}$, Lemma 2 guarantees that $C' \cup (C \setminus C') \in [\![FM_2]\!]^{ow}$. Observing that $C' \cup (C \setminus C') = C' \cup C$ and $C' \cup C = C$ because $C' \subseteq C$ by construction of C', we can conclude $C \in [\![FM_2]\!]^{wc}$. □

The above enables the definition of a simple algorithm for checking whether the open-world semantics of a FM FM_1 is included in the open-world semantics of a FM FM_2 based iteratively enumerating all configurations of features that are used in FM_1 or FM_2, which are finitely many. Theorem 1 guarantees that there exist semantic differences from FM_1 to FM_2 under the open-world semantics if, and only if, one of these configurations is valid in FM_1 and not valid in FM_2. More efficient implementations that exploit the optimizations implemented in modern SAT-solvers are also possible: Similar to the method for semantic FM differencing under the closed-world semantics [2], for two FMs FM_1 and FM_2, we can construct two formulas Φ_1 and Φ_2 with the following properties: The formula Φ_1 contains a variable for each feature in FM_1. The formula Φ_2 contains a variable for each feature in FM_2. Each model of Φ_1 encodes a valid configuration of FM_1 and each model of Φ_2 encodes a valid configuration of FM_2. If each feature that is used in both FMs is encoded by the same variable in both formulas, then each model of the formula $\Phi_1 \wedge \neg\Phi_2$ encodes a valid configuration of FM_1 that is no valid configuration of FM_2. Theorem 1 guarantees that there are semantic differences from FM_1 to FM_2 under the open-world semantics if, and only if, $\Phi_1 \wedge \neg\Phi_2$ is satisfiable. Similarly, semantic differencing of the FMs under the closed-world semantics is possible via checking whether the following formula is satisfiable [2]: $(\Phi_1 \wedge (\bigwedge_{f \in F_2 \setminus F_1} \neg x_f)) \wedge \neg(\Phi_2 \wedge (\bigwedge_{f \in F_1 \setminus F_2} \neg x_f))$ where F_1 is the set of features of FM_1, F_2 is the set of features of FM_2, and x_f is the variable for feature f.

5 DISCUSSION: SEMANTICS IN DIFFERENT DEVELOPMENT PHASES

Feature oriented development processes [9, 25] divide into two phases, *i.e., domain engineering* in which "commonality and variability of the product line are defined and realised" [9], and *application engineering* "in which the applications of the product line are built by reusing domain artefacts and exploiting the product line variability" [9]. Semantic differencing of FMs facilitates to analyze how the range of products in the product line evolves. Independent of the semantics, incomparable semantics of two consecutive versions of a FM indicate that the new version excludes configurations from the original version and also adds configurations that were not possible in the original version.

During the domain engineering phase, features and their relationships are identified and added to FMs for obtaining a complete model of the product line. Open-world semantic differencing facilitates to compare consecutive versions of the FMs during domain engineering following the intuition that adding features actually refines the set of configurations described by the FM (cf. Section 4). During the *domain engineering phase*, product line specifications are created, *i.e.,* features and their relationships are identified and added to a FM. Therefore, this phase usually underlies the assumption that the FM under development does not already contain all relevant



	FM1		FM2		FM3
$[\![\ ]\!]^{cw}$	$\{\emptyset, \{B\}\}$	$\not\supseteq$ $\not\subseteq$	$\{\emptyset, \{A\}\}$	$\subseteq$	$\{\emptyset, \{A\}, \{A, B\}\}$
$[\![\ ]\!]^{ow}$	all configurations	$\subseteq$ $\supseteq$	all configurations	$\supseteq$	$\{C \mid B \in C \Rightarrow A \in C\}$

Figure 6: Refinement using one of the semantics does not imply refinement using the other semantics.

features. Therefore, in this phase, the addition of information is usually considered to be a refinement in the sense that the addition of information excludes elements from the model's semantics. Thus, using the open-world semantics usually yields the expected results for semantic differencing in the domain engineering phase.

During the *application engineering phase*, developers *exploit* (use) the FM created during the domain engineering phase. The individual features are realized and choosing a product of the FM composes the realizations of the product's features to obtain a realization of the product. In this phase, changing a FM such that it permits new products, *e.g.,* by adding features, effectively changes the realizations of the product line's products and might even cause already implemented products to become invalid. Each newly added product should be detected and reviewed by product line engineers to ensure that the product line only permits meaningful realizations. Thus, in the application engineering phase, semantic differencing should report every newly added product when the set of products of a FM changes, *e.g.,* when features are added. Therefore, semantic differencing using the closed-world semantics [2] yields the appropriate results in this development phase. Refinement of FMs in the application engineering phase corresponds to further restricting the already existing product range by excluding redundant products or products that are technically impossible to implement. Therefore, for ensuring that products are not removed unintentionally during the application engineering phase, the semantic difference from an original FM version to a successor version should detect all products that have been removed during the evolution to the successor version. This is also achievable with the closed-world semantics.

Semantic differencing is a valuable operation in software product line development, where the expected result depends on the semantics definition. Using the open-world semantics in early and the closed-world semantics in late development phases usually meets the purposes of the different development phases and the intuition of developers. Therefore, semantic FM evolution analysis by means of semantic differencing usually requires to adapt the semantics definition according to the development phase.

6 RELATED AND AFFECTED WORK

Existing approaches for semantic FM differencing [2, 11] are based on the closed-world semantics. As discussed in Section 4 and Section 3, using the open-world semantics may change the result when conducting semantic differencing significantly. The appropriate semantics used for semantic differencing depends on the analyst's intuition, which usually differs in early and late development phases (cf. Section 5). Semantic differencing is only one out of many automated analyses for FMs (*e.g.,* [6, 42, 47, 53]). To the best of our knowledge, all semantic analyses of FMs are based on the closed-world semantics. In general, all of these analyses are also applicable using the open-world semantics. However, using the open-world semantics might change the perspective of the analyses in the sense that they provide other information as the analyses using the closed-world semantics.

This section presents related work on automated FM analyses, composition, and synthesis. For each of the approaches, we discuss the differences from the existing method's results under the closed-world semantics to the results under the open-world semantics.

6.1 Translations for Automated Reasoning

There are well-known translations from FMs to propositional formulas such that the interpretations satisfying the formula obtained from translating a FM represent exactly the products of the FM [2, 6, 8, 16, 55]. As presented in Section 4, these translations are easily adaptable for semantic FM differencing using the open-world semantics. Similarly, all reasoning approaches on FMs based on propositional logic using, *e.g.,* binary decision diagrams [10] as proposed in [7], can be applied for open-world semantic analyses by adapting the translation to propositional logic. Description logic is another formalism used for automated reasoning on FMs [6, 54]. The approach [54] assumes closed-world semantics and can be similarly adapted as the approaches using propositional formulas to base the analyses on the open-world semantics. Furthermore, automatic reasoning on FMs has been conducted using constraint programming [50]. The rules for mapping a FM to a constraint satisfaction problem [8] can equally be adapted by excluding the constraints, which state that features have to be contained in the FM's set of features.

6.2 Semantic Property Analyses

There are automatic analyses for the plausibility of FMs in terms of properties of their semantics. As these analyses are based on the semantics of FMs, the analysis results change when using the open-world semantics instead of the closed-world semantics.

Dead Feature Detection. A feature is dead in a FM, if it is not part of any of the FM's modeled configurations [6]. With the closed-world semantics, a feature is dead if it is not modeled in the FM or if it is constrained by a cross-tree constraints such that it cannot be part of any modeled configuration [6]. Under the open-world semantics, a feature not modeled in a FM is not dead. In case the feature is modeled in the FM, it is dead under the closed-world semantics, if it is dead under the open-world semantics.

Verifying Products or Partial Configurations. Operators for verifying products check whether a product is valid in the modeled product line [6]. As defined in Section 3, every product of a FM (element of the closed-world semantics) is also valid in the FM (element of the open-world semantics). Choosing the appropriate semantics depends on the intuition of the product line engineer when using the operator (cf. Section 5). A similar operation is the verification of partial configurations [6]. Valid partial configurations must not contain contradictions with respect to the cross-tree constraints modeled in the FM. Again, the result of applying the operator depends on the chosen semantics and the appropriate semantics depends on the developer's intuition as well as the current development phase.

Feature Model Satisfiability. A FM is satisfiable, if it permits at least one configuration, *i.e.,* its semantics contains at least one element that is different from the empty configuration [6]. If a FM is not satisfiable, then it contains contradicting cross-tree constraints [6]. There are well-known automated methods for checking whether a FM is satisfiable under the closed-world semantics [5, 6, 24, 48], which are, *e.g.,* based on checking the satisfiability of the formula resulting from translating the FM to propositional logic. The approaches are reusable for consistency checking using the open-world semantics, if the universe of features is finite and the propositional formula resulting from the translation is interpreted over the variables encoded by the elements of the finite universe. If the universe of features is infinite, then every FM is satisfiable under the open-world semantics (cf. Lemma 4). This reflects the suitability of the open-world semantics in the domain engineering phase, where feature constraints are yet to be identified.

Corrective Explanations. A corrective explanation provides information that explain why an operator yields its result. This facilitates developers in finding deficiencies that should be corrected. Examples for corrective explanations have been proposed, *e.g.,* by [49] introducing abductive reasoning or by [11, 26, 34] for analyzing the syntactic changes, which lead to some semantic relation (such as refinement) between two FMs. For instance, [26] presents a language independent theory to obtain a sequence of change operations to repair a failed model refinement. The formal approach is exemplary applied to FMs using the closed-world semantics. Each of the approaches [11, 26, 34, 49] bases its semantic FM analyses on the propositional formulas obtained from translating FMs. The analyses of the approaches are adaptable to the open-world semantics by interchanging the translation from FMs to propositional formulas as discussed above.

Product enumeration methods. Operators to determine the number of specified products [8, 15] or even the entire range of possible products aim at revealing the level of variability within the product line and to identify possible extensions of the initially intended product scope [6]. Feature dependency analyses [39] are used to identify undesirable dependencies between features within a FM. Using the open-world semantics, the results of these analyzes are usually (assuming an infinite universe of features) infinite sets that do not provide valuable information. Furthermore, these operators analyze the variability or deficiencies within a modeled product line that is assumed to be almost complete. Therefore, applying these analyses is not meaningful in early development phases, where many features of the domain of interest may be missing in the FM.

6.3 Semantics-aware Feature Model Composition and Synthesis

Composition and synthesis of FMs are valuable operations that target FM reuse in software product line development.

Feature Model Composition. There are various composition operators for FMs [1, 2, 43, 51, 52]. Each composition operator combines two FMs to obtain another FM that satisfies certain properties with respect to the original FMs' semantics. For instance, an intersection composition operator takes two FMs as input and returns another FM with a semantics that is the intersection or a superset of the intersection of the semantics of the input FMs [1, 2, 43, 51]. Analogously, a FM union composition operator takes two FMs as input and returns a new FM with a semantics that is the union or a superset of the union of the input FMs' semantics [1, 2, 43, 52]. The existing composition operators consider the closed-world semantics. The relationship between the existing composition operators and the open-world semantics reveals reuse potential: Assume reusing an intersection composition operator that composes two FMs that share the same features such that the resulting FM has the same features as the input FMs and the resulting FM's closed-world semantics is equal to the intersection of the input FMs' closed-world semantics. Then, the resulting FM's open-world semantics is equal to the intersection of the input FMs' open-world semantics.

Lemma 6. *Let FM_1, FM_2, FM_3 be feature models that all share the same set of features. If $[\![FM_3]\!]^{cw} = [\![FM_1]\!]^{cw} \cap [\![FM_2]\!]^{cw}$, then $[\![FM_3]\!]^{ow} = [\![FM_1]\!]^{ow} \cap [\![FM_2]\!]^{ow}$.*

Proof. Let FM_1, FM_2, FM_3 be three feature models sharing the same features such that $[\![FM_3]\!]^{cw} = [\![FM_1]\!]^{cw} \cap [\![FM_2]\!]^{cw}$. Let F denote the set of features of the three FMs.

"$\subseteq$": Let $c \in [\![FM_3]\!]^{ow}$ be a valid configuration of FM_3. By Lemma 1, we have that $c \cap F \in [\![FM_3]\!]^{ow}$ is also valid in FM_3. As $c \cap F$ is valid in FM_3 and $(c \cap F) \subseteq F$, we obtain with Definition 4 that $c \cap F \in [\![FM_3]\!]^{cw}$. From this, as by assumption it holds that $[\![FM_3]\!]^{cw} = [\![FM_1]\!]^{cw} \cap [\![FM_2]\!]^{cw}$, we can infer that $c \cap F \in [\![FM_1]\!]^{cw}$ and $c \cap F \in [\![FM_2]\!]^{cw}$. Therefore, with Lemma 3, we obtain $c \cap F \in [\![FM_1]\!]^{ow}$ and $c \cap F \in [\![FM_2]\!]^{ow}$. With this and as $c \setminus F \subseteq U_F \setminus F$, we can conclude with Lemma 2 that $c = (c \cap F) \cup (c \setminus F) \in [\![FM_1]\!]^{ow}$ and $c = (c \cap F) \cup (c \setminus F) \in [\![FM_2]\!]^{ow}$. Therefore, $c \in [\![FM_1]\!]^{ow} \cap [\![FM_2]\!]^{ow}$.

"$\supseteq$": Let $c \in [\![FM_1]\!]^{ow} \cap [\![FM_2]\!]^{ow}$ be a configuration that is valid in FM_1 and in FM_2. By Lemma 1, we have that $c \cap F \in [\![FM_1]\!]^{ow} \cap [\![FM_2]\!]^{ow}$ is also valid in FM_1 and in FM_2. As $c \cap F$ is valid in FM_1 and in FM_2 and as $c \cap F \subseteq F$, we obtain with Definition 4 that $c \cap F \in [\![FM_1]\!]^{cw} \cap [\![FM_2]\!]^{cw}$. Using the assumption $[\![FM_3]\!]^{cw} = [\![FM_1]\!]^{cw} \cap [\![FM_2]\!]^{cw}$, we obtain that $c \cap F \in [\![FM_3]\!]^{cw}$. With Lemma 3, this implies $c \cap F \in [\![FM_3]\!]^{ow}$. With this and as $c \setminus F \subseteq U_F \setminus F$, we can conclude with Lemma 2 that $c = (c \cap F) \cup (c \setminus F) \in [\![FM_3]\!]^{ow}$. □

Reusing union composition operators yields similar results, *i.e.,* the resulting FM's open-world semantics is equal to the union of the input FMs' open-world semantics.

Lemma 7. *Let FM_1, FM_2, FM_3 be feature models that all share the same set of features. If $[\![FM_3]\!]^{cw} = [\![FM_1]\!]^{cw} \cup [\![FM_2]\!]^{cw}$, then $[\![FM_3]\!]^{ow} = [\![FM_1]\!]^{ow} \cup [\![FM_2]\!]^{ow}$.*

Proof. Let FM_1, FM_2, FM_3 be three feature models sharing the same features such that $[\![FM_3]\!]^{cw} = [\![FM_1]\!]^{cw} \cup [\![FM_2]\!]^{cw}$. Let F denote the set of features of the three FMs.

"$\subseteq$": Let $c \in [\![FM_3]\!]^{ow}$ be a valid configuration of FM_3. By Lemma 1, we have that $c \cap F \in [\![FM_3]\!]^{ow}$ is also valid in FM_3. As $c \cap F$ is valid in FM_3 and $(c \cap F) \subseteq F$, we obtain with Definition 4 that $c \cap F \in [\![FM_3]\!]^{cw}$. From this, as by assumption it holds that $[\![FM_3]\!]^{cw} = [\![FM_1]\!]^{cw} \cup [\![FM_2]\!]^{cw}$, we can infer that $c \cap F \in [\![FM_1]\!]^{cw}$ or $c \cap F \in [\![FM_2]\!]^{cw}$. Therefore, with Lemma 3, we obtain $c \cap F \in [\![FM_1]\!]^{ow}$ or $c \cap F \in [\![FM_2]\!]^{ow}$. With this and as $c \setminus F \subseteq U_F \setminus F$, we can conclude with Lemma 2 that $c = (c \cap F) \cup (c \setminus F) \in [\![FM_1]\!]^{ow}$ or $c = (c \cap F) \cup (c \setminus F) \in [\![FM_2]\!]^{ow}$. Therefore, $c \in [\![FM_1]\!]^{ow} \cup [\![FM_2]\!]^{ow}$.

"$\supseteq$": Let $c \in [\![FM_1]\!]^{ow} \cup [\![FM_2]\!]^{ow}$ be a configuration that is valid in FM_1 or in FM_2. By Lemma 1, we have that $c \cap F \in [\![FM_1]\!]^{ow} \cup [\![FM_2]\!]^{ow}$ is also valid in FM_1 or in FM_2. As $c \cap F$ is valid in FM_1 or in FM_2 and as $c \cap F \subseteq F$, we obtain with Definition 4 that $c \cap F \in [\![FM_1]\!]^{cw} \cup [\![FM_2]\!]^{cw}$. Using the assumption $[\![FM_3]\!]^{cw} = [\![FM_1]\!]^{cw} \cup [\![FM_2]\!]^{cw}$, we obtain that $c \cap F \in [\![FM_3]\!]^{cw}$. With Lemma 3, this implies $c \cap F \in [\![FM_3]\!]^{ow}$. With this and as $c \setminus F \subseteq U_F \setminus F$, we can conclude with Lemma 2 that $c = (c \cap F) \cup (c \setminus F) \in [\![FM_3]\!]^{ow}$. □

Feature Model Synthesis. The FM synthesis problem [16] takes a propositional formula over features as input. The idea is to compute a FM such that its closed-world semantics is equal to the satisfying interpretations of the formula. Concerning the open-world semantics, the translation constructs a FM such that all the constraints induced by the formula must be respected by the FM's valid configurations. This also implies that every configuration over the FM's features that is no product of the resulting FM is also no valid configuration of the resulting FM. Existing FM synthesis methods are thus well reusable when using the open-world semantics.

7 CONCLUSION AND FUTURE PROSPECTS

Semantic differencing of FMs facilitates to analyze how the range of products in the product line evolves. Using approaches for semantic differencing based on the open-world semantics, additionally to those based on the closed-world semantics, supports model evolution analysis of product refinement, especially in the design phase. Our semantic differencing operator for FMs using the open-world semantics facilitates the comparison of FMs for product line managers during early development stages. The proposed open-world semantics is especially useful in early development stages or when using agile development, where requirements are permanently subject to change. Automatic semantic differencing of FMs using the open-world semantics is possible as it suffices to search all possible configurations that solely contain features that exist in the input FMs. As argued in Section 5, using both, open-world semantics in early development stages, and closed-world semantics in late development stages, provides developers the best of both worlds. For the early development stages, it is important that the addition of information to a model causes that the resulting model is a refinement of its predecessor version.

The integration of the semantics in various tools is subject to future work. For developers, a graphical representation of valid configurations in the compared FMs could be useful to facilitate the understanding of the differences. Moreover, it will be interesting to provide a tool, which integrates the different approaches for syntactic and semantic differencing of FMs and provides developers a comfortable way to make their analyses.

REFERENCES

[1] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. 2010. Composing Feature Models. In *Software Language Engineering.*

[2] Mathieu Acher, Patrick Heymans, Philippe Lahire, Clément Quinton, Philippe Collet, and Philippe Merle. 2012. Feature Model Differences. In *Advanced Information Systems Engineering - 24th International Conference.*

[3] Marcus Alanen and Ivan Porres. 2003. Difference and Union of Models. In *«UML» 2003 - The Unified Modeling Language. Modeling Languages and Applications.*

[4] Sven Apel and Christian Kästner. 2009. An Overview of Feature-Oriented Software Development. *Journal of Object Technology* 8, 5 (2009), 49–84.

[5] Don Batory. 2005. Feature Models, Grammars, and Propositional Formulas. In *Software Product Lines.*

[6] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Information Systems* 35, 6 (2010), 615–636.

[7] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. 2007. FAMA: Tooling a Framework for the Automated Analysis of Feature Models. In *Proceeding of the First International Workshop on Variability Modelling of Softwareintensive Systems (VAMOS).*

[8] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. 2005. Automated Reasoning on Feature Models. In *Proceedings of the 17th International Conference on Advanced Information Systems Engineering.*

[9] Böckle, Günter and Pohl, Klaus and van der Linden, Frank. 2005. A Framework for Software Product Line Engineering. In *Software Product Line Engineering: Foundations, Principles, and Techniques.* Springer Berlin Heidelberg, 19–38.

[10] Randal E. Bryant. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Comput.* 35, 8 (1986), 677–691.

[11] Johannes Bürdek, Timo Kehrer, Malte Lochau, Dennis Reuling, Udo Kelter, and Andy Schürr. 2016. Reasoning About Product-line Evolution Using Complex Feature Model Differences. *Automated Software Engineering* 23, 4 (2016), 687–733.

[12] Arvid Butting, Oliver Kautz, Bernhard Rumpe, and Andreas Wortmann. 2017. Semantic Differencing for Message-Driven Component & Connector Architectures. In *International Conference on Software Architecture.*

[13] Rafael Capilla, Jan Bosch, Pablo Trinidad, Antonio Ruiz-Cortés, and Mike Hinchey. 2014. An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry. *Journal of Systems and Software* 91 (2014).

[14] Paul Clements and Linda Northrop. 2007. *Software Product Lines: Practices and Patterns.* Addison-Wesley.

[15] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. 2005. Formalizing Cardinality-based Feature Models and their Specialization. In *Software Process: Improvement and Practice*, Vol. 10. 7–29.

[16] Krzysztof Czarnecki and Andrzej Wasowski. 2007. Feature Diagrams and Logics: There and Back Again. In *Software Product Line Conference (SPLC 2007).*

[17] Ferruccio Damiani, Luca Padovani, and Ina Schaefer. 2012. A Formal Foundation for Dynamic Delta-oriented Software Product Lines. In *Proceedings of the 11th International Conference on Generative Programming and Component Engineering.*

[18] Uli Fahrenberg, Mathieu Acher, Axel Legay, and Andrzej Wąsowski. 2014. Sound Merging and Differencing for Class Diagrams. In *Fundamental Approaches to Software Engineering.*

[19] Uli Fahrenberg, Axel Legay, and Andrzej Wąsowski. 2011. Vision Paper: Make a Difference! (Semantically). In *Model Driven Engineering Languages and Systems.*

[20] Robert France and Bernhard Rumpe. 2007. Model-Driven Development of Complex Software: A Research Roadmap. In *Future of Software Engineering 2007 at ICSE.*

[21] Christian Gerth, Jochen M. Küster, Markus Luckey, and Gregor Engels. 2010. Precise Detection of Conflicting Change Operations Using Process Model Terms. In *Model Driven Engineering Languages and Systems.*

[22] Hans Grönniger, Holger Krahn, Claas Pinkernell, and Bernhard Rumpe. 2008. Modeling Variants of Automotive Systems using Views. In *Proceedings of Workshop Modellbasierte Entwicklung von eingebetteten Fahrzeugfunktionen (MBEFF).*

[23] Christoph Herrmann, Holger Krahn, Bernhard Rumpe, Martin Schindler, and Steven Völkel. 2007. An Algebraic View on the Semantics of Model Composition. In *Model Driven Architecture- Foundations and Applications.*

[24] Kyo Kang, Sholom Cohen, James Hess, William Nowak, and Spencer Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* Technical

Report. Software Engineering Institute - Carnegie Mellon University.

[25] Kang, Kyo C. and Kim, Sajoong and Lee, Jaejoon and Kim, Kijoo and Shin, Euiseob and Huh, Moonhang. 1998. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. *Annals of Software Engineering* 5, 1.

[26] Oliver Kautz and Bernhard Rumpe. 2018. On Computing Instructions to Repair Failed Model Refinements. In *Conference on Model Driven Engineering Languages and Systems (MODELS'18)*.

[27] Philipp Kehrbusch, Johannes Richenhagen, Bernhard Rumpe, Axel Schloßer, and Christoph Schulze. 2016. Interface-based Similarity Analysis of Software Components for the Automotive Industry. In *International Systems and Software Product Line Conference (SPLC '16)*.

[28] Timo Kehrer, Udo Kelter, and Gabriele Taentzer. 2011. A Rule-Based Approach to the Semantic Lifting of Model Differences in the Context of Model Versioning. In *International Conference on Automated Software Engineering (ASE'11)*.

[29] Timo Kehrer, Udo Kelter, and Gabriele Taentzer. 2013. Consistency-Preserving Edit Scripts in Model Versioning. In *International Conference on Automated Software Engineering (ASE)*.

[30] Jochen M. Küster, Christian Gerth, and Gregor Engels. 2009. Dependent and Conflicting Change Operations of Process Models. In *Model Driven Architecture - Foundations and Applications*.

[31] Jochen M. Küster, Christian Gerth, Alexander Förster, and Gregor Engels. 2008. Detecting and Resolving Process Model Differences in the Absence of a Change Log. In *Business Process Management*.

[32] Philip Langer, Tanja Mayerhofer, and Gerti Kappel. 2014. A Generic Framework for Realizing Semantic Model Differencing Operators. In *PSRC@MoDELs (CEUR Workshop Proceedings)*, Vol. 1258. CEUR-WS.org.

[33] Philip Langer, Tanja Mayerhofer, and Gerti Kappel. 2014. Semantic Model Differencing Utilizing Behavioral Semantics Specifications. In *Model-Driven Engineering Languages and Systems*.

[34] Shahar Maoz and Jan Oliver Ringert. 2016. A framework for relating syntactic and semantic model differences. *Software & System Modeling* 17, 3 (2016).

[35] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. 2010. A Manifesto for Semantic Model Differencing. In *Proceedings Int. Workshop on Models and Evolution (ME'10) (LNCS 6627)*. Springer, 194–203.

[36] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. 2011. ADDiff: Semantic Differencing for Activity Diagrams. In *Conference on Foundations of Software Engineering (ESEC/FSE '11)*. ACM, 179–189.

[37] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. 2011. CDDiff: Semantic Differencing for Class Diagrams. In *ECOOP 2011 – Object-Oriented Programming*.

[38] Tanja Mayerhofer, Philip Langer, and Gerti Kappel. 2015. Semantic Model Differencing Based on Execution Traces. In *Software Engineering & Management*.

[39] Marcílio Mendonça, Donald Cowan, William Malyk, and Toacy Oliveira. 2008. Collaborative Product Configuration: Formalization and Efficient Algorithms for Dependency Analysis. *Journal of Software* 3, 2 (2008).

[40] Klaus Pohl, Günter Böckle, and Frank van der Linden. 2005. *Software Product Line Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[41] Raymond Reiter. 1978. On Closed World Data Bases. In *Logic and Data Bases*. Springer US.

[42] Camille Salinesi and Raúl Mazo. 2012. Defects in Product Line Models and how to Identify them. In *Software Product Line - Advanced Topic*. InTech editions.

[43] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. 2007. Generic Semantics of Feature Diagrams. *Computer Networks* 51, 2 (2007).

[44] Mirjam Steger, Christian Tischer, Birgit Boss, Andreas Müller, Oliver Pertler, Wolfgang Stolz, and Stefan Ferber. 2004. Introducing PLA at Bosch Gasoline Systems: Experiences and Practices. In *Software Product Line Conference (SPLC'04)*.

[45] Gabriele Taentzer, Claudia Ermel, Philip Langer, and Manuel Wimmer. 2014. A fundamental approach to model versioning based on graph modifications: from theory to implementation. *Software & Systems Modeling* 13, 1 (2014).

[46] Thomas Thüm, Don Batory, and Christian Kästner. 2009. Reasoning about Edits to Feature Models. In *Proceedings of the 31st International Conference on Software Engineering*.

[47] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. 2014. FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming* 79 (2014).

[48] Pablo Trinidad, David Benavides, Amador Durán, Antonio Ruiz-Cortés, and Miguel Toro. 2008. Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software* 81, 6 (2008).

[49] Pablo Trinidad and Antonio Ruiz Cortés. 2009. Abductive Reasoning and Automated Analysis of Feature Models: How are they connected?. In *Third International Workshop on Variability Modelling of Software-Intensive Systems*.

[50] Edward Tsang. 1993. *Foundations of Constraint Satisfaction*. Academic Press.

[51] Pim van den Broek. 2012. Intersection of Feature Models. In *Software Product Line Conference (SPLC'12)*.

[52] Pim van den Broek, Ismênia Galvão, and Joost Noppen. 2010. Merging Feature Models. In *Software Product Line Conference (SPLC'10)*.

[53] Thomas von der Maßen and Horst Lichter. 2004. Deficiencies in Feature Models. In *Workshop on Software Variability Management for Product Derication*.

[54] Hai Wang, Yuan Fang Li, Jing Sun, Hongyu Zhang, and Jeff Pan. 2005. A Semantic Web Approach to Feature Modeling and Verification. In *Workshop on Semantic Web Enabled Software Engineering (SWESE'05)*.

[55] Wei Zhang, Haiyan Zhao, and Hong Mei. 2004. A Propositional Logic-Based Method for Verification of Feature Models. In *Formal Methods and Software Engineering (FSE'04)*. Springer Berlin Heidelberg.