Scientific Annals of Computer Science vol. 26 (1), 2016, pp. 69–122
doi: 10.7561/SACS.2016.1.69

Faithful Modeling of Product Lines with Kripke Structures and Modal Logic

Zinovy Diskin[1,2], Aliakbar Safilian[1], Tom Maibaum[1], Shoham Ben-David[2]

Abstract

Software product lines are now an established framework for software design. They are specified by special diagrams called feature models. For formal analysis, the latter are usually encoded by Boolean propositional theories. We discuss a major deficiency of this semantics, and show that it can be fixed by considering a product to be an instantiation process rather than its final result. We call intermediate states of this process partial products, and argue that what a feature model really defines is a poset of its partial products. We argue that such structures can be viewed as special Kripke structure that we call *partial product Kripke structures, ppKS*. To specify these Kripke structures, we propose a CTL-based logic, called *partial product CTL, ppCTL*. We show how to represent a feature model M by a ppCTL theory $\mathsf{ML}(M)$ (ML stands for modal logic) such that any ppKS satisfying the theory is equal to the partial product line determined by M. Hence, $\mathsf{ML}(M)$ can be considered a sound and complete representation of M. We also discuss several applications of the modal logic view in feature modeling, including refactoring of feature models.

Keywords: software product lines, feature models, partial product Kripke structures, partial product lines, partial product CTL, faithful semantics

[1]Department of Computing and Software, McMaster University, E-mails: {`disnkinz, safiliaa, maibaum`}`@mcmaster.ca`

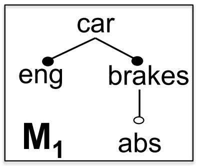
[2]Department of Electrical and Computer Engineering, University of Waterloo, E-mails: `zdiskin@gsd.uwaterloo.ca, shohambd@gmail.com`

`Names of the first two authors are written in the alphabetical order.`

1 Introduction

The *software product line* approach is well-known in the software industry. Products in a product line (PL) share some common *mandatory* features, and differ by having some *optional* features that allow the user (or developer) to configure the product the user wants (e.g., MS Office, a Photoshop, or the Linux kernel). Instead of producing a multitude of separate products, the vendor designs a single PL encompassing a variety of products, which results in a significant reduction in development time and cost [32]. Methods of specifying PLs and checking the validity of a PL against a specification is an active research area.

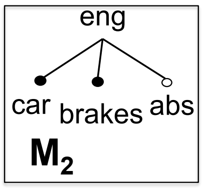
The most common method for designing a PL is to build a *feature model* (FM); below we will often say just model. A toy example is shown in the inset figure. Model M_1 says that a (root feature called) car *must* have an engine and brakes (black bullets denote *mandatory* subfeatures), and brakes can *optionally* (note the hollow bullet) be equipped with an anti-skidding system. The model specifies a PL consisting of two products: $P = \{\mathsf{car}, \mathsf{eng}, \mathsf{brakes}\}$ and $P' = P \cup \{\mathsf{abs}\}$.



As industrial models may be based on thousands of features inter-related in complex ways [27], they require tools for their management and analysis, and thus should be represented by formal objects processable by tools. A common approach is to consider features as atomic propositions, and view a model as a theory in the Boolean propositional logic (BL), whose valid valuations are to be exactly the valid products defined by the model [3]. For example, model M_1 represents the BL theory (i.e., a set of Boolean propositional formulas) $\mathsf{BL}(M_1) = \{\mathsf{car}\} \cup \{\mathsf{eng}{\rightarrow}\mathsf{car}, \mathsf{brakes}{\rightarrow}\mathsf{car}, \mathsf{abs}{\rightarrow}\mathsf{brakes}\} \cup \{\mathsf{car}{\rightarrow}\mathsf{eng}, \mathsf{car}{\rightarrow}\mathsf{brakes}\}$: the first three implications encode subfeature dependencies (a feature can appear in a product only if its parent is in the product), the last two implications encode the mandatory dependencies between features (if a parent of a mandatory feature is included in the product, then it must included too), and the root feature must be always included in the product. We call this semantics of models *Boolean*.
The Boolean semantics gave rise to a series of prominent applications for analysis of industrial size PLs [14, 19, 37]. However, it has an almost evident drawback of misrepresenting models' hierarchical structure. Indeed, the second inset figure shows a model M_2 that is essentially different from M_1 (and is, in fact, pathological), but has the same set of products, $\{P, P'\}$, determined by an equivalent Boolean theory $\mathsf{BL}(M_2) = \{\mathsf{car}{\rightarrow}\mathsf{eng}, \mathsf{brakes}{\rightarrow}\mathsf{eng},$

$\mathsf{abs}{\to}\mathsf{eng}\}{\cup}\{\mathsf{eng}{\to}\mathsf{car}, \mathsf{eng}{\to}\mathsf{brakes}\} \cong \mathsf{BL}(M_1)$: only grouping of implications has changed, but it is immaterial for Boolean logic. The core of the problem is that two different dependencies (the parent feature and a mandatory subfeature) are similarly encoded by implication, and hence are not semantically distinguished.

We are not the first to have noticed this drawback, e.g., it is mentioned in [37] (where models' semantics not captured by Boolean logic is called *ontological*), and many researchers and practitioners in the field are probably aware of the situation. Nevertheless, as far as we know, no alternative to the Boolean logic semantics of feature modeling has been proposed in the literature, which we think is theoretically and conceptually unsatisfactory. Even more importantly, inadequate logical foundations for feature modeling hinder practical analyses: as important information contained in models is not captured by their BL-encoding, this information is either missing from analyses, or treated informally, or hacked in an ad hoc way. In a sense, this is yet another instance of a known software engineering problem, when semantics is hidden in the application code rather than explicated in the specification, with all its negative consequences for software testing, debugging, maintenance, and communication between the stakeholders.



Our main observation is that the key notion of feature modeling—a product built from features—should be considered as an *instantiation process* rather than its final result. We call intermediate states of this process *partial products*, and argue that what a model M really specifies is a partially ordered set of partial products, which we call a *partial product line* (PPL) generated by M. The commonly considered products of M (we call them *full*) only form a subset of M's PPL. We then show that any PPL can be viewed as an instance of a special type of Kripke structures, which we axiomatically define and call a *partial product Kripke structure* (ppKS). The latter are specifiable by a suitable version of modal logic, which we call *partial product CTL* (ppCTL), as it is basically a fragment of CTL enriched with a constant modality that only holds in states representing full products. We show that any model M can be represented by a ppCTL (modal) theory $\mathsf{ML}(M)$ accurately specifying M's intended semantics: for any ppKS K, $K \models \mathsf{ML}(M)$ iff K is equal to M's PPL, and hence $\mathsf{ML}(M)$ is a *sound* and *complete* representation of the model. Then we can replace models by the respective ppCTL-theories, which are well amenable to formal analysis and automated processing.

In a broader perspective, we want to show that behavioural foundations of feature modeling are mathematically interesting, and worth the attention of the Theoretical Computer Science community. We will describe several open problems that we believe are mathematically interesting and practically useful. Especially intriguing are connections to concurrency modeling. In fact, PLs can be seen as a special interpretation of configuration structures [40]: features are events, partial products are configurations, and PPLs are configuration structures; feature models can then be seen as a far reaching generalization of Winskel's event structures and other formalisms for specifying dependencies between events. It appears that the syntactical aspects of specifying concurrency (including transaction mechanisms), i.e., having a convenient and suggestive notation suitable for practitioners, have not received much attention in concurrency modeling. This is where we believe feature modeling can make a non-trivial contribution. We will discuss some details in Sect. 8.1. On the other hand, we would like to have the paper readable by a feature modeling researcher, and to convince her that the logic of models is modal rather than Boolean. Therefore, we pay special attention to the motivation of our framework: we want first to validate the mathematical framework, and only then explore it formally.

This paper extends our previous shorter paper [16]. Here we provide the proofs of our results that have been omitted in [16]: we explicate the structure of modal theories extracted from models, and present detailed proofs following this structure. Moreover, we discuss bisimulation and simulation relations on ppKSs. We also discuss refactoring of feature models in the entirely new Sect. 6, and show that the notion of PPL, i.e., ppKS semantics for feature models, captures not only constraints embodied in models, but their feature hierarchy as well (the latter was always a challenging issue for the Boolean semantics [37]). Therefore we called the ppKS semantics *faithful*. We have also added an analysis of a special version of ppKS semantics, in which the i2c-principle is not assumed. The paper also provides a more complete review of the related work, and extends the future work section.

Our plan for the paper is as follows. Section 2 is motivational: we describe the basics of feature modeling, and show how the deficiency of the Boolean semantics can be fixed by introducing *partial* products and transitions between them. In Sect. 3, the notions of models and PPLs they generate are formalized. In Sect. 4, we introduce the notion of ppKSs as immediate abstraction of PPLs, and ppCTL as a language to specify ppKSs' properties. In this section, we also discuss bisimulation and simulation relations on ppKSs and show that they are equal to identity and substructure relations over ppKSs, respectively. We show, step-by-step, how to translate a model into a ppCTL-theory, and prove our soundness and completeness theorems in Sect. 5. Sect. 6 discusses the notion of refactoring of feature models by using bisimulation relation on their PPLs. An important result of the study in this section is that PPLs are faithful to the semantics of models. In Sect. 7, we discuss some practical applications. Related work is discussed in Sect. 8, and future work in Sect. 9. Section 10 concludes. Appendices A.1 and A.2 respectively show complete BL and ppCTL encodings of our running example in Fig. 1.

2 Feature Models and Partial Product Lines

This section aims to motivate the formal framework we develop in the paper. In Sect. 2.1, we discuss the basics of feature modeling, and in Sect. 2.2 we introduce partial products and PPLs. We begin with PPLs generated by simple models, which can be readily explained in lattice-theoretic terms. Then (Sect. 2.3) we show that PPLs generated by complex models are more naturally, and even necessarily, to be considered as transition systems.

2.1 Basics of Feature Modeling

A model is a graphical structure presenting a hierarchical decomposition of features with some possible *crosscutting constraints* (CCs) between them. Figure 1 gives an example. It is a tree of features, whose root names the product ('car' in this case), and edges relate a feature to its subfeatures. Edges with black bullets denote *mandatory* subfeatures: every car *must* have an eng (engine), a gear, and brakes. The hollow-ended edge says that brakes can *optionally* be equipped with abs. Black angles denote *OR-groups*: an engine can be either gas (gasoline), or elec (electric), or both. Hollow angles denote *XOR-groups* (eXclusive OR): a gear is either mnl (manual) or atm (automatic) but not both; it must be supplied with oil as dictated by the black-bullet edge. The $\times$-ended arc says that an electric engine cannot be combined with a manual gear, and the arrow-headed arc says that an automatic gear requires ABS. According to the model, the set of features {car, eng, gas, gear, mnl, oil, brakes} is a valid product, but replacing the gasoline engine by electric, or removal of oil, would make the product invalid.

In this way, the model compactly specifies seven valid products amongst the set of 2^9 possible combinations of 9 non-root features (the root is always included), and exhibits dependencies between choices.

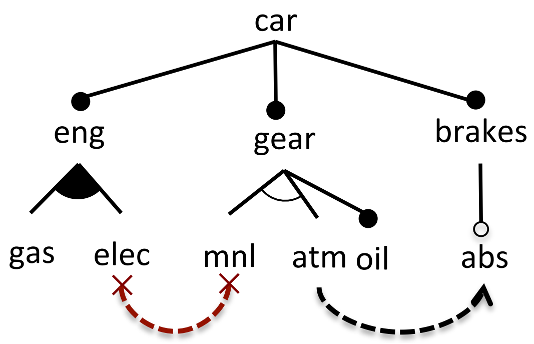


Figure 1: An model

In the Boolean view of feature modeling, a model is a representation of a BL theory. For example, the theory encoded by the model in Fig. 1 consists of a set of implications denoting subfeature dependencies and unary mandatory dependencies as explained in the introduction, plus three implications denoting grouped mandatoriness: $\{\mathsf{eng}\to\mathsf{gas}\vee\mathsf{elec},\ \mathsf{gear}\to\mathsf{mnl}\vee\mathsf{atm},\ \mathsf{mnl}\wedge\mathsf{atm}\to\bot\}$ (with $\bot$ denoting False), plus two implications encoding CCs: $\{\mathsf{elec}\wedge\mathsf{mnl}\to\bot, \mathsf{atm}\to\mathsf{abs}\}$. Since the root must be always included in a valid product, we also add the theory car. However, as we saw in the Introduction, a BL encoding is deficient.

2.2 Partial Product Lines: Products as Processes

What is lost in the BL-encoding is the *dynamic* nature of the notion of products. A model defines not just a set of valid products but the very way these products are to be (dis)assembled step by step from constituent features. Correspondingly, a PL appears as a transition system initialized at the root feature (say, car for model M_1 in Fig. 2a) and gradually progressing towards fuller products (say, $\{\mathsf{car}\}\to\{\mathsf{car},\mathsf{eng}\}\to\{\mathsf{car},\mathsf{eng},\mathsf{brakes}\}$ or $\{\mathsf{car}\}\to\{\mathsf{car},\mathsf{brakes}\}\to\{\mathsf{car},\mathsf{brakes},\mathsf{abs}\}\to\{\mathsf{car},\mathsf{brakes},\mathsf{abs},\mathsf{eng}\}$); we call such sequences *instantiation paths.*

The graph in Fig. 2(b1) specifies all possible instantiation paths for M_1 (c, e, b, a stand for car, eng, brakes, abs, resp., to make the figure compact). Nodes in the graph denote *partial* products, i.e., valid products

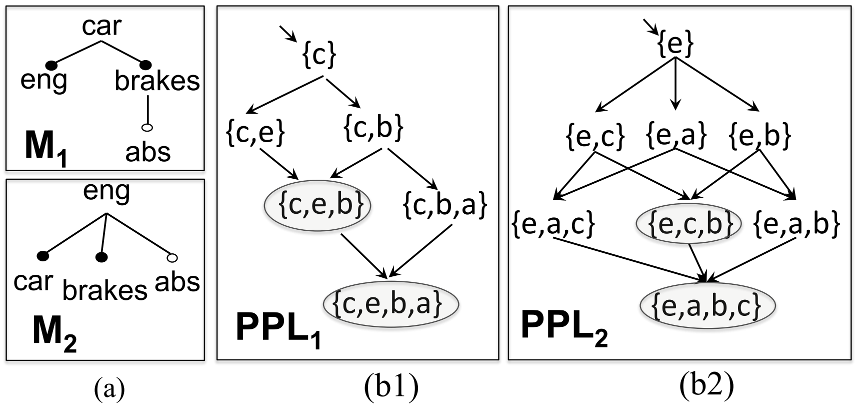


Figure 2: From models to PPLs: simple cases

with, perhaps, some mandatory features missing: for example, product {c,e} is missing feature b, and product {c,b} is missing feature e. In contrast, products {e} and {c,a} are invalid as they contain a feature without its parent; such products do not occur in the graph. As a rule, we will call partial products just *products*. Product {c,e,b} is *full* (complete) as it has all mandatory subfeatures of its member-features; nodes denoting full products are framed. (Note that product {c,e,b} is full but not terminal, whereas the bottom product is both full and terminal.) Edges in the graph denote inclusions between products. Each edge encodes adding a single feature to the product at the source of the edge; in text, we will often denote such edges by an arrow and write, e.g., $\{c\} \longrightarrow_e \{c, e\}$, where the subscript denotes the added feature. We call the instantiation graph described above the *partial product line* determined by model M_1, and write PPL_1. In a similar way, the PPL of the second model, PPL_2, is built in Fig. 2(b2). We see that although both models have the same set of *full* products (i.e., are Boolean semantics equivalent), their PPLs are essentially different both structurally (6 nodes and 7 edges in PPL_1 versus 8 nodes and 12 edges in PPL_2), and in the content of products (e.g., products {c} and {c,b} present in PPL_1 but absent in PPL_2, whereas {e} and {e,a} are present in PPL_2 but absent from PPL_1). This essential difference between PPLs properly reflects the essential difference between the models.

2.3 Partial Product Lines: From lattices to transition systems

Generating PPLs $PPL_{1,2}$ from models $M_{1,2}$ in Fig. 2 can be readily explained in lattice-theoretic terms. Let us first forget about mandatory bullets, and consider all features as optional. Then both models are just trees, and hence are posets, even join semi-lattices (joins go up in feature trees). Valid products of model M_i are upward-closed sets of features (filters), and form a lattice (consider Fig. 2(b1,b2) as Hasse diagrams), whose join is set union, and meet is intersection. If we freely add meets (go down) to posets $M_{1,2}$ ($\mathsf{eng} \wedge \mathsf{brakes}$ etc.), and thus freely generate lattices $L(M_i)$, $i = 1, 2$, over the respective posets, then lattices $L(M_i)$ and PPL_i will be dually isomorphic (Birkhoff duality).

The forgotten mandatoriness of some features appears as incompleteness of some objects; we call them *proper* partial products. Partial products closed under mandatoriness are *full*. Thus, PPLs of simple models such as in Fig. 2(a) are their filter lattices with distinguished subsets of full products. In the next section, we will argue that this lattice-theoretic view does not work for more complex models.

Figure 3 (left) shows a fragment of the model in Fig. 1, in which, for uniformity, we have presented the XOR-group as an OR-group with a new CC added to the tree (note the $\times$-ended arc between mnl and atm)[3]. To build the PPL, we follow the idea described above, and first consider M_3 as a pure tree-based poset with all the extra-structure (denoted by black bullets and black triangles) removed. Figure 3 (right) describes a part of the filter lattice as a Hasse diagram (ignore the difference between solid and dashed edges for a while); to ease reading, the number of letters in the acronym for a feature corresponds to its level in the tree, e.g., c stands for car, en for eng etc.

Now let us consider how the additional structure embodied in the model influences the PPL. Two CCs force us to exclude the bottom central and right products from the PPL; they are shown in brown-red and the respective edges are dashed. To specify this lattice-theoretically, we add to the lattice of features a universal *bottom* element $\perp$ (a feature to be a subfeature of any feature), and write two defining equations: $\mathsf{ele} \wedge \mathsf{mnl} = \perp$ and $\mathsf{mnl} \wedge \mathsf{atm} = \perp$. Then, in the filter lattice, the formal down-join of products {c,en,ele,ge} and {c,ge,mnl,en} "blow up" and become equal to the set of all features ("False

[3]Recall that an $\times$-ended arc between two incomparable features denotes an exclusive constraint CC between them.

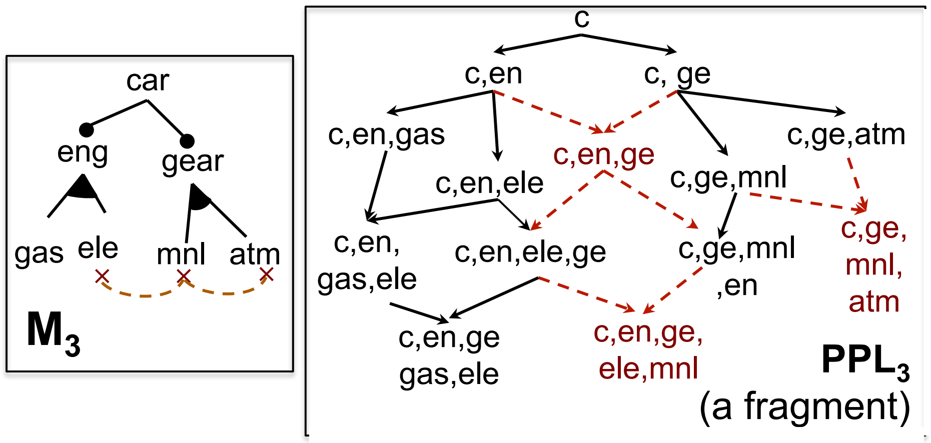


Figure 3: From models to PPLs: Complex case

implies everything"). The same happens with the other pair of conflicting products.

Next we consider the mandatoriness structure of M_3 (given by black bullets and triangles). This structure determines a subset of the PPL consisting of full products (not shown in Fig. 3) as we discussed above. In addition, mandatoriness affects the set of valid partial products as well. Consider the product $P = \{\mathsf{c}, \mathsf{en}, \mathsf{ge}\}$ at the center of the diagram. The left instantiation path leading to this product, $\{\mathsf{c}\} \longrightarrow_{\mathsf{en}} \{\mathsf{c}, \mathsf{en}\} \longrightarrow_{\mathsf{ge}} P$ is not good because gear was added to engine *before* the latter is fully assembled (a mandatory choice between being electric or gasoline, or both, has still not been made). Jumping to another branch from inside of the branch being processed can be considered a poor design practice that the modeler may want to prohibit by declaring the corresponding transition as invalid. Then transition $\{\mathsf{c}, \mathsf{ge}\} \longrightarrow_{\mathsf{en}} P$ should be also invalid as engine is added before gear instantiation is completed. Hence, product P becomes unreachable, and should be removed from the PPL. (In the diagram, invalid edges are dashed (red with a color display), and the products at the ends of such edges are invalid too.)

Thus, a reasonable requirement for the instantiation process is that processing a new branch of the feature tree should only begin after processing of the current branch has reached a full product. We call this requirement *instantiate-to-completion* (i2c) by analogy with the *run-to-completion* transaction mechanism in behavior modeling (indeed, instantiating a branch of a feature tree can be seen as a transaction).

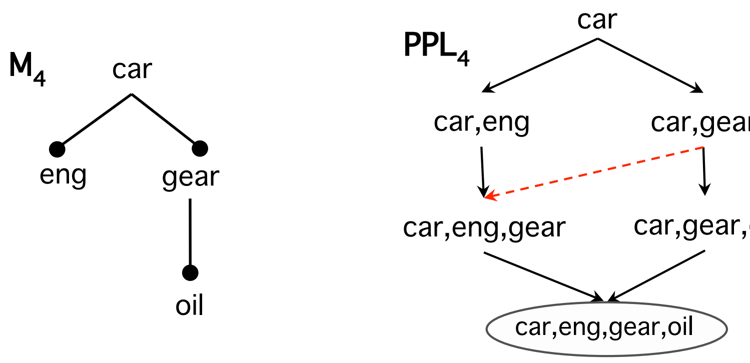


Figure 4: Exclusion of an edge due to i2c

Importantly, i2c prohibits transitions rather than products, and it is possible to have a product with some instantiation paths into it being legal (and hence the product is legal as well), but some paths to the product being illegal. Figure 4 shows a simple example with model M_4 and its PPL. In the latter, the "diagonal" transition $\{\mathsf{car}, \mathsf{gear}\} \longrightarrow \{\mathsf{car}, \mathsf{eng}, \mathsf{gear}\}$ violates i2c and must be removed. However, its target product is still reachable from $\{\mathsf{car}, \mathsf{eng}\}$ as the latter is a fully instantiated product. Hence, the only element excluded by i2c is the diagonal dashed transition.

Note that the i2c principle may substantially reduce the complexity of the PPL for a given model, as it may excludes many transitions and/or states from the PPL, without loss of any information of the model. It follows from this observation that a PPL can be richer than its lattice of partial products (transition exclusion cannot be explained lattice-theoretically), and transition systems/Kripke structures and modal logic are needed.

Moreover, even if all inclusions are transitions, Boolean logic is too poor to express important semantic properties embodied in PPLs. For example, we may want to say that every product can be completed to a full product, and every full product is a result of such a completion. Or, we may want to say that if a product P has some feature f, then in some of its partial completions P', a feature g should appear. Or, if a product P has a feature f, then any full product completing P must have a feature g, and so on. Specification of such properties needs some version of modal logic. In general, since modal logic is more expressive than Boolean, it provides a more expressive language for cross-cutting constraints over feature models. Later in Sect. 7, we will provide an example, in which some practically reasonable constraints cannot be expressed in Boolean logic and require a modal specification.

Thus, the transition relation is an important (and independent) component of the general PPL structure. As soon as transitions become first-class citizens, it makes sense to distinguish full products by supplying them, and only them, with identity loops. That is, each framed product in our figures describing PPLs, should be assumed to have a loop transition to itself. Such loops do not add (nor remove) any feature from the product, and have a clear semantic meaning: the instantiation process can stay in a full product state indefinitely. This way, the transition relation in a PPL would be left-total,[4] which makes PPLs standard Kripke structures used for the semantics of CTL, in which transition relations must be left-total.

3 Feature models and their PPLs: formally

In Sect. 3.1, we give a formal definition of a (feature) model that supports all our work in the paper. Sect. 3.2 defines a Boolean logic encoding of a model, and the corresponding notions of a full and a partial products. Sect. 3.3 formally defines a PPL as a transition systems.

3.1 Feature Models

Several versions of feature models and their Boolean semantics are uniformly formalized in [36]. We develop yet another formalization of tree-based models as a quadruple of components, which is basically equivalent to the above, but our choice of the components provides feature models with a structure that supports all our work in the paper. Particularly, this structure is important for translating models into Boolean and modal theories, and for specifying their relationships and refactoring, and hence facilitating reverse engineering of models from product lines. Finally, we will need this structure in our future work on feature model management, for which we will need to define morphisms between models

Typical models are trees of features with some extra structures, like in Fig. 1. In our framework, mandatory features and XOR-groups are derived constructs. A mandatory feature can be seen as a singleton OR-group. An XOR-group can be expressed by an OR-group with some additional exclusive constraints between its elements.

[4]A relation $R \subseteq A \times B$ is left-total if $\forall a \in A, \exists b \in B\colon (a,b) \in R$

Definition 1 (Feature Diagrams) A *feature diagram* (FD) is a pair $T_{\mathcal{OR}} = (T, \mathcal{OR})$ of the following components.

(i) $T = (F, r, \_^{\uparrow})$ is a tree whose nodes are *features*: F denotes the set of all features, $r \in F$ is the root, and function $\_^{\uparrow}$ maps each non-root feature $f \in F_{-r} \overset{\text{def}}{=} F \setminus \{r\}$ to its parent $f^{\uparrow}$. The inverse function that assigns to each feature the set of its children (called *subfeatures*) is denoted by $f_{\downarrow}$; this set is empty for leaves. It is easy to see that the set of f's siblings is the set $(f^{\uparrow})_{\downarrow} \setminus \{f\}$. The set of all ancestors and all descendants of a feature f are denoted by $f^{\uparrow\uparrow}$ and $f_{\downarrow\downarrow}$, respectively.

Features f, g are called *incomparable*, $f\#g$, if neither of them is a descendant of the other. We write $\#2^F$ for the set $\{G \subset F : G \neq \emptyset \text{ and } f\#g \text{ for all } f, g \in G\} \subset 2^F$.

(ii) $\mathcal{OR}$ is a function that assigns to each feature $f \in F$ a set $\mathcal{OR}(f) \subset 2^{f_{\downarrow}}$ (possibly empty) of *disjoint* subsets of f's children called *OR-groups*. If a group $G \in \mathcal{OR}(f)$ is a singleton $\{f'\}$ for some $f' \in f_{\downarrow}$, we say that f' is a *mandatory* subfeature of f. For example, in Fig. 1, $\mathcal{OR}(\mathsf{gear}) = \{\{\mathsf{mnl}, \mathsf{aut}\}, \{\mathsf{oil}\}\}$, and oil is a mandatory subfeature of gear.

Elements in set $O(f) \overset{\text{def}}{=} f_{\downarrow} \setminus \bigcup \mathcal{OR}(f)$ are called *optional* subfeatures of f. For example, in Fig. 1, $\mathcal{OR}(\mathsf{brakes}) = \varnothing$, and abs is an optional subfeature of brakes. □

An model is a feature diagram plus some possible exclusive and/or inclusive crosscutting constraints:

Definition 2 (Feature Models) A *feature model* (model) is a triple $M = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ with $T_{\mathcal{OR}}$ a feature diagram as defined above, and two additional components defined below:
(i) $\mathcal{EX} \subseteq \#2^F$ is a set of *exclusive dependencies* between features. For example, in Fig. 1, $\mathcal{EX} = \{\{\mathsf{elec}, \mathsf{mnl}\}, \{\mathsf{mnl}, \mathsf{atm}\}\}$.
(ii) $\mathcal{IN} \subset \#2^F \times \#2^F$ is a set of *inclusive dependencies* between features. A member of this set is interpreted (and written) as an implication $(f_1 \wedge \ldots \wedge f_m) \to (g_1 \vee \ldots \vee g_n)$. For example, feature model in Fig. 1 has $\mathcal{IN} = \{\mathsf{atm} \to \mathsf{abs}\}$.

Exclusive and inclusive dependencies are also called *cross-cutting constraints* (CCs).[5] □

Thus, an model is a tree of features T endowed with three extra structures $\mathcal{OR}$, $\mathcal{EX}$, and $\mathcal{IN}$. We will sometimes write it as a quadru-

[5]It is easy to see that any Boolean constraint/formula can be expressed as a conjunction of our $\mathcal{EX}$ and $\mathcal{IN}$ dependencies.

ple $M = (T, \mathcal{OR}, \mathcal{EX}, \mathcal{IN})$. If needed, we will subscript M's components with index $_M$, e.g., write F_M for the set of features F. Note that an model is a purely syntactic object contrary to the common usage of term 'model' in logic. The class of all feature models over the same set of features F is denoted by $\mathbf{FM}(F)$.

3.2 Propositional encoding of models

We will first present the general idea in Sect. 3.2.1, then show how to modify it to manage the major drawback of the standard Boolean encoding in Sect. 3.2.2, and finally discuss a propositional encoding of $\mathsf{i2c}$ in Sect. 3.2.3

3.2.1 The approach

A common approach to formalizing the PL (of full products) of a given model is to use Boolean propositional logic [3]. Features are considered as atomic propositions, and dependencies between features are specified by logical formulas. For example, if a feature f' is a subfeature of feature f, we have an implication $f' \to f$ (if a product has feature f', it must have feature f as well). If $\{g_1, g_2\}$ is an OR-group of f's subfeatures, we write $f \to (g_1 \vee g_2)$; if, in addition, features g_1, g_2 are mutually exclusive, we write $g_1 \wedge g_2 \to \bot$. In this way, given a model $M = (T, \mathcal{OR}, \mathcal{EX}, \mathcal{IN})$, each of its four components gives rise to a respective propositional theory (i.e.,a set of formulas) as shown in the upper four rows of Table 1: later we will discuss the four theories in detail and explain the !-superscripts.[6]

Together these theories constitute theory $\mathsf{BL}^!(M)$, and a set of features P is a legal full product for M iff $P \models \mathsf{BL}^!(M)$. Here $\models$ denotes the standard satisfaction relation between a set of atomic propositions (features in our context) P and a Boolean theory Ψ: we define $P \models \Psi$ iff $P \models \psi$ for all $\psi \in \Psi$, and for a Boolean formula $\psi = \psi(f_1, \ldots, f_n)$ built over atomic propositions f_1,...f_n, we define $P \models \psi$ iff $\psi(\bar{f}_1, \ldots, \bar{f}_n) = 1$, where $\bar{f}_i = 1$ for $f_i \in P$ and $\bar{f}_i = 0$ otherwise.[7]

Since publishing the seminal paper [27], this propositional view of feature modeling became common and has been used in both theoretical

[6] $\bigvee G$ and $\bigwedge G$ represent the conjunction and the disjunction of all formulas in a set of formulas G.

[7] Later we will also need semantic consequence between theories, $\Psi_1 \models \Psi_2$, which means $P \models \Psi_2$ for any $P \models \Psi_1$. Also note that $P \models \Psi$ is equivalent to the universal validity of the formula $\bigwedge\{p_1, \ldots, p_n, \neg q_1, \ldots \neg q_k\} \to \bigwedge \Psi$, where propositions q_j are all those that do not occur into P, and $\neg q_j$ denotes negation of q_j.

and practice-oriented work [3,14,37].

Table 1: Boolean theories extracted from a model $M = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$

(1)	$\mathsf{BL}(T) = \{\top \to r\} \cup \{f' \to f\colon\ f \in F, f' \in f_{\downarrow}\}$
(2)	$\mathsf{BL}(\mathcal{EX}) = \{\bigwedge G \to \bot\colon\ G \in \mathcal{EX}\}$
($3^!$)	$\mathsf{BL}^!(\mathcal{OR}) = \{f \to \bigvee G\colon\ f \in F, G \in \mathcal{OR}(f)\}$
($1{+}3^!$)	$\mathsf{BL}^!(T_{\mathcal{OR}}) = \mathsf{BL}(T) \cup \mathsf{BL}^!(\mathcal{OR})$
($4^!$)	$\mathsf{BL}^!(\mathcal{IN}) = \{\bigwedge G \to \bigvee G'\colon\ (G, G') \in \mathcal{IN}\}$
($\text{all}^!$)	$\mathsf{BL}^!(M) = \mathsf{BL}(T) \cup \mathsf{BL}(\mathcal{EX}) \cup \mathsf{BL}^!(\mathcal{OR}) \cup \mathsf{BL}^!(\mathcal{IN})$
(3)	$\mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}}) = \big\{f \wedge g \to (\bigwedge \mathsf{BL}^!(T^f_{\mathcal{OR}})) \vee (\bigwedge \mathsf{BL}^!(T^g_{\mathcal{OR}}))\ :\ f^{\uparrow} = g^{\uparrow}\big\}$
(all)	$\mathsf{BL}(M) = \mathsf{BL}(T) \cup \mathsf{BL}(\mathcal{EX}) \cup \mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}})$

3.2.2 Enabling vs. Causality, or Full vs. Partial Products

The encoding above has a drawback that we discussed in the Introduction: two different relationships between features (being a subfeature, $f' \to f$, and being a mandatory subfeature, $f \to f'$) are similarly encoded. This implies $f \leftrightarrow f'$ for any mandatory subfeature f' of f, and leads to misrepresentation of the hierarchical structure of a model. With a more refined approach, the two relationships should be represented differently.

The subfeature relationship is fundamental, and any product having a subfeature f' but missing its superfeature f should be considered ill-formed; we can say that superfeature f *enables* its subfeature f' and all reasonable products must respect enabling. In contrast, if f' is a mandatory subfeature of f, a product having f but missing f' is just incomplete rather than ill-formed. We can say that feature f *causes* f' so that partial products violating causality are possible, and only full products must respect it.[8]

Thus, we have two Boolean theories for the same model M. One is the theory of partial products and another is the theory of full products. The theory of *partial products* is denoted by $\mathsf{BL}(M)$ (for now without the bang

[8]Our choice of terms 'enabling' and 'causal' for the two types of structural dependencies is somewhat arbitrary, and was partly motivated by similarities between feature and event modeling discussed later in Sect. 8.1.

superscript): it encodes the basic structural dependencies a well-formed partial product must satisfy, and thus defines all partial products. This theory consists of three components as specified in row (all) in Table 1: $\mathsf{BL}(T)$ is the BL-encoding of subfeature dependencies (row (1)), $\mathsf{BL}(\mathcal{EX})$ is the BL-encoding of exclusive dependencies (row (2)), and in Sect. 3.2.3 we consider yet another ingredient—the BL encoding of the i2c-condition, $\mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}})$.

The other propositional theory, M's *full product theory* $\mathsf{BL}^{!}(M)$, consists of four components. Two components are $\mathsf{BL}(T)$ and $\mathsf{BL}(\mathcal{EX})$ as above, the third one is the BL-encoding $\mathsf{BL}^{!}(\mathcal{OR})$ of the mandatoriness dependencies embodied in the $\mathcal{OR}$-structure (row ($3^{!}$)), and the fourth is a Boolean encoding $\mathsf{BL}^{!}(\mathcal{IN})$ of the inclusive crosscutting constraints (row($4^{!}$)), which we treat as mandatory for only full products rather than affecting instantiation (i.e., as causal rather than enabling). We also consider the theory $\mathsf{BL}^{!}(T_{\mathcal{OR}})$ as the union of $\mathsf{BL}(T)$ and $\mathsf{BL}^{!}(\mathcal{OR})$. With a more refined approach to feature modeling, a crosscutting constraint should be labeled as either causal or enabling, but with the current feature modeling practice, crosscutting constraints are not labeled and we thus consider them as causal, i.e., constraining full products only.

Definition 3 (Full Products) A *full product* over an model $M = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ is a set of features $P \subseteq F$ satisfying theory $\mathsf{BL}^{!}(M)$ defined in Table 1 in row ($\text{all}^{!}$). The set of all full products is called M's *full product set* and denoted by $\mathcal{FP}_M$. Thus, $\mathcal{FP}_M = \{P \subseteq F : P \models \mathsf{BL}^{!}(M)\}$. □

The definition above is equivalent to the standard one, except that we use the term *full* product rather than product. To define partial products, we need to introduce one more ingredient of the instantiation theory.

3.2.3 Instantiate to Completion and Transient Conflicts.

Consider once again PPL$_3$ in Fig. 3, from which product $\{\mathsf{c}, \mathsf{en}, \mathsf{ge}\}$ is excluded as violating the i2c principle. Note that in order to specify this exclusion propositionally, we *cannot* declare that features en and ge are mutually exclusive and write $\{\mathsf{en} \wedge \mathsf{ge} \to \bot\}$ because further down the lattice they are combined in product $\{\mathsf{c}, \mathsf{en}, \mathsf{ele}, \mathsf{ge}\}$ below $\{\mathsf{c}, \mathsf{en}\}$, and in product $\{\mathsf{c}, \mathsf{ge}, \mathsf{mnl}, \mathsf{en}\}$ below $\{\mathsf{c}, \mathsf{ge}\}$ as well. In other words, the conflict between features en and ge is transient rather than permanent, and its propositional specification is not trivial. We solve this problem as follows.

Definition 4 (Induced Subtrees) Let $T_{\mathcal{OR}} = (T, \mathcal{OR})$ be a feature diagram over a set of features F, and $f \in F$. A *feature subtree induced* by f is a pair $T^f_{\mathcal{OR}} = (T^f, \mathcal{OR}^f)$ with T^f being the tree under f, i.e., $T^f \overset{\mathrm{def}}{=} (f_{\downarrow\downarrow} \cup \{f\}, f, \_^{\uparrow})$, and mapping $\mathcal{OR}^f$ is inherited from $\mathcal{OR}$, i.e., for any $g \in f_{\downarrow\downarrow}$, $\mathcal{OR}^f(g) = \mathcal{OR}(g)$. □

The theory formalizes the idea that if a valid product contains two incomparable features, then at least one of these features must be fully instantiated within the product. Now we can specify theory $\mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}})$ as shown in row (3) in Table 1.

Definition 5 (Partial Products) A *partial product* over model $M = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ is a set of features $P \subseteq F$ satisfying the instantiation theory $\mathsf{BL}(M)$ specified in row (all) in Table 1. (Recall that a full product is a set of features satisfying theory $\mathsf{BL}^!(M)$.) We denote the set of all partial products by $\mathcal{PP}(M)$ or sometimes $\mathcal{PP}_M$. Thus, $\mathcal{PP}(M) = \{P \subseteq F : P \models \mathsf{BL}(M)\}$.

We will often call partial products just products. □

The following proposition is obvious.

Proposition 1 For any model M and any product P, $P \vDash \mathsf{BL}^!(M) \Rightarrow P \vDash \mathsf{BL}(M)$, i.e., $\mathsf{BL}^!(M) \models \mathsf{BL}(M)$. Hence, full products as defined in Definition 3 form a subset of partial products, $\mathcal{FP}(M) \subseteq \mathcal{PP}(M)$. □

Appendix A.1 represents the Boolean logic theory of the whole model in Fig. 1. Note that transition exclusion discussed in Sect. 2.3 cannot be explained with Boolean logic and needs a modal logic; we will give a suitable logic and show how it works in Sect. 5.

3.3 PPLs as Transition Systems

In this section, we consider how partial products are related. The problem we address is when a valid product P can be augmented with a feature $f \notin P$ so that product $P' = P \cup \{f\}$ is valid as well. We then write $P \longrightarrow P'$ and call the pair (P, P') *a valid* (*elementary* or *step*) *transition.*

Two necessary conditions are obvious: the parent $f^{\uparrow}$ must be in P, and f should not be in conflict with features in P, that is, $P' \models \mathsf{BL}(T) \cup \mathsf{BL}(\mathcal{EX})$. Compatibility with $\mathsf{i2c}$ is more complicated: we need to formalize *relative* completeness of P in its branch.

Definition 6 (Relative fullness) Given a product P and a feature $f \notin P$, the following theory (continuing the list in Table 1) is defined:

$$(3)_{P,f} \qquad \mathsf{BL}^{\mathsf{i2c}}(P,f) \overset{\text{def}}{=} \bigcup \{\mathsf{BL}^{!}(T^{g}_{\mathcal{OR}}) : g \in P \cap (f^{\uparrow})_{\downarrow}\}$$

where $T^{g}_{\mathcal{OR}}$ denotes the subtree induced by feature g as described in Definition 4. (Note that set $P \cap (f^{\uparrow})_{\downarrow}$ may be empty, and then theory $\mathsf{BL}^{\mathsf{i2c}}(P,f)$ is also empty.) We say P is *fully instantiated with respect to* f if $P \models \mathsf{BL}^{\mathsf{i2c}}(P,f)$. $\square$

For example, it is easy to check that for model M_4 in Fig. 4, for product $P_1 = \{\mathsf{car}, \mathsf{eng}\}$ and feature $f_1 = \mathsf{gear}$, we have $P_1 \models \mathsf{BL}^{\mathsf{i2c}}(P_1, f_1)$ while for $P_2 = \{\mathsf{car}, \mathsf{gear}\}$ and $f_2 = \mathsf{eng}$, $P_2 \not\models \mathsf{BL}^{\mathsf{i2c}}(P_2, f_2)$ because $\mathsf{BL}^{!}(T^{\mathsf{gear}}_{\mathcal{OR}}) = \{\mathsf{gear} \rightarrow \mathsf{oil}\}$ and $P_2 \not\models \{\mathsf{gear} \rightarrow \mathsf{oil}\}$.

Now, we are at the point where we can give a formal definition for valid transitions:

Definition 7 (Valid Transitions) Let P be a product. Pair (P, P') is a *valid transition*, we write $P \longrightarrow P'$, iff one of the following two possibilities (a), (b) holds.

(a) $P' = P \uplus \{f\}$ for some feature $f \notin P$ such that the following three conditions hold: (a1) $P' \models \mathsf{BL}(T)$, (a2) $P' \models \mathsf{BL}(\mathcal{EX})$, and (a3) $P \models \mathsf{BL}^{\mathsf{i2c}}(P,f)$.

(b) $P' = P$ and P is a full product.

That is, $P \longrightarrow P'$ iff $\big((a1) \wedge (a2) \wedge (a3)\big) \vee (b)$. $\square$

For example, the dashed (red) transition in Fig. 4 is not valid because $P = \{\mathsf{car}, \mathsf{gear}\} \not\models \mathsf{BL}^{\mathsf{i2c}}(P, \mathsf{eng})$. The following result is important.

Theorem 1 If P is a valid partial product and $P \longrightarrow P'$, then P' is a valid partial product.

Proof: If $P' = P$, the proposition is obvious. Consider now the case of $P' = P \uplus \{f\}$ with $P' \models \mathsf{BL}(T) \cup \mathsf{BL}(\mathcal{EX})$ and $P \models \mathsf{BL}^{\mathsf{i2c}}(P,f)$. We need to prove that $P' \models \mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}})$. Let $g \in P$ be an arbitrary feature with $g^{\uparrow} = f^{\uparrow}$, i.e., $g \in P \cap (f^{\uparrow})_{\downarrow}$. By definition of relative fullness, if $P \models \mathsf{BL}^{\mathsf{i2c}}(P,f)$, then definitely $P \models \mathsf{BL}^{!}(T^{g}_{\mathcal{OR}})$ (one of the union's components). This implies $P' \models \mathsf{BL}^{!}(T^{g}_{\mathcal{OR}})$, and hence $P' \models \bigcup \big\{\mathsf{BL}^{!}(T^{g}_{\mathcal{OR}}) \colon g \in P, g^{\uparrow} = f^{\uparrow})\big\}$. The above statement, along with $P \models \mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}})$, implies that $P' \models \big\{f \wedge g \rightarrow (\bigwedge \mathsf{BL}^{!}(T^{f}_{\mathcal{OR}})) \vee (\bigwedge \mathsf{BL}^{!}(T^{g}_{\mathcal{OR}}))$ $\square$

Finally, we formalize PPLs as follows.

Definition 8 (Partial Product Lines) Let $M = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ be an model. The *partial product line* (PPL) determined by M is a triple $\mathbb{P}(M) = (\mathcal{PP}_M, \longrightarrow_M, I_M)$ with the set $\mathcal{PP}_M$ of partial products given by Definition 5, transition relations $\longrightarrow_M$ given by Definition 7 (so that full products, and only them, are equipped with self-loops), and the initial product $I_M = \{r\}$ consisting of the root feature. $\square$

Below in Sect. 6, we show that PPLs provide a *faithful* semantics for models, which captures both the products and the tree-structure of feature models (see Discussion on page 101).

4 Partial Product Kripke Structures

In this section, we introduce *partial product Kripke structures*, which are an immediate abstraction of PPLs generated by models. We then discuss simulation and bisimulation relations on these special Kripke structures.

By *Kripke structures*, we understand a family of mathematical structures of the following format. We first fix a set A of atomic propositions, and then consider a tuple $K = (W, R, L)$ with W a set of (*possible*) *worlds* or *states.* R a binary *transition* relation over W, and L a labelling function $W \to 2^A$, which maps a world to the set of propositions true in this world. Partial product lines motivate a specialization of the notion, in which worlds (called partial products) are identified with sets of atomic propositions (features), and hence labelling is not needed. Full products are identified by loops on corresponding states. These structures also satisfy some special properties defined in the following definition.

Definition 9 (partial product Kripke Structure) Let F be a finite set (of *features*). A *partial product Kripke structure* (ppKS) over F is a triple $K = (\mathcal{PP}, \longrightarrow, I)$ with $\mathcal{PP} \subset 2^F$ a set of non-empty *(partial) products*, $I \in \mathcal{PP}$ the *initial* singleton product (i.e., $I = \{r\}$ for some $r \in F$), and $\longrightarrow \subseteq \mathcal{PP} \times \mathcal{PP}$ a binary left-total *transition* relation[9]. In addition, the following three conditions hold ($\longrightarrow^+$ denotes the transitive closure of $\longrightarrow$):
(*Singletonicity*) For all $P, P' \in \mathcal{PP}$, if $P \longrightarrow P'$ and $P \neq P'$, then $P'{=}P \uplus \{f\}$ for some $f \notin P$.
(*Reachability*) For all $P \in \mathcal{PP}$, $I \longrightarrow^+ P$, i.e., P is reachable from I.
(*Self-Loops Only*) For all $P, P' \in \mathcal{PP}$, if ($P \longrightarrow^+ P' \longrightarrow^+ P$), then $P = P'$, i.e., every loop is a self-loop.

[9] A binary relation R over a set A is called left-total if $\forall a \in A, \exists b \in A : R(a, b)$.

A product P with $P \longrightarrow P$ is called *full*. The set of full products is denoted by $\mathcal{FP}$. □

The components of a ppKS K are subscripted with $_K$ if needed, e.g., $\mathcal{PP}_K$. We denote the class of all ppKSs built over a set of features F by $\mathbf{KS}(F)$. Note that any partial product in a ppKS eventually evolves into a full product because F is finite, $\longrightarrow$ is left-total, and all loops are self-loops. It means that any ppKS enjoys the following property
(*Finality*) For all $P \in \mathcal{PP}$, there exists a full product P' such that $P \longrightarrow^* P'$, where $\longrightarrow^*$ denotes the reflexive transitive closure of $\longrightarrow$.

The following statement is an obvious corollary of Definition 8.

Corollary 1 Let $M \in \mathbf{FM}(F)$ be a model. Its partial product line is an ppKS, i.e., $\mathbb{P}(M) \in \mathbf{KS}(F)$. □

We will also need the notion of sub-ppKS.

Definition 10 (Sub-ppKS) Let K and K' be two ppKSs. We say K is a sub-ppKS of K', denoted by $K \preceq_{\text{sub}} K'$, iff $\mathcal{PP}_K \subseteq \mathcal{PP}_{K'}$, and $\longrightarrow_K \subseteq \longrightarrow_{K'}$. □

It is easy to see that $K \preceq_{\text{sub}} K'$ implies $I_K = I_{K'}$.

Consider two Kripke structures, K and K', which are respectively built over the set of atomic propositions A and A' such that $A \subseteq A'$. A relation R from the states of K to the states of K' is called a *simulation* [7] if for any states s and s', $s\ R\ s'$ if the label of s is a subset of the label of s' and for any transition $s \longrightarrow t$ in K there is a transition $s' \longrightarrow t'$ in K' such that $t\ R\ t'$. We say that K' *simulates* K and write $K \preceq_{\text{sim}} K'$ if there is a simulation relation R from K to K' such that $s_0\ R\ s'_0$, where s_0 and s'_0 are the initial states in K and K', respectively. We say that two Kripke structures K and K' are *simulation equivalent* if $K \preceq_{\text{sim}} K'$ and $K' \preceq_{\text{sim}} K$. The following theorem shows that the restriction of the simulation relation on ppKSs is equal to the substructure relation on them.

Theorem 2 Given two ppKSs, $K \in \mathbf{KS}(F)$ and $K' \in \mathbf{KS}(F')$ with $F \subset F'$, $K \preceq_{\text{sim}} K'$ iff $K \preceq_{\text{sub}} K'$.

Proof. Consider two ppKSs, K and K', as above. We will first show that
($\Rightarrow$) $K \preceq_{\text{sim}} K'$ implies $\mathcal{PP}_K \subseteq \mathcal{PP}_{K'}$ and $\longrightarrow_K \subseteq \longrightarrow_{K'}$.
Suppose that $K \preceq_{\text{sim}} K'$ via a simulation relation $R \subseteq \mathcal{PP}_K \times \mathcal{PP}_{K'}$. Since $R(I_K, I_{K'})$ and both $I_K, I_{K'}$ are singletons, $I_K = I_{K'}$. In particular,

$\mathcal{PP}_K \cap \mathcal{PP}_{K'} \neq \varnothing$.
Let $P \in \mathcal{PP}_K \cap \mathcal{PP}_{K'}$ such that $R(P, P')$. Consider a transition $P \longrightarrow_K Q$ (in K). Since $K \preceq_{\text{sim}} K'$, there exists a transition $P \longrightarrow_{K'} Q'$ (in K') such that $R(Q, Q')$. Due to the singletonicity condition of ppKSs (see Definition 9), there are $f \in F, f' \in F'$ such that $f, f' \notin P$, $Q = P \cup \{f\}$, and $Q' = P \cup \{f'\}$. Since $Q \subseteq Q'$ (as $R(Q, Q')$) and $f, f' \notin P$, f must be equal to f', which implies that $Q = Q'$. Hence, for any $P \in \mathcal{PP}_K \cap \mathcal{PP}_{K'}$, the following holds:
(*) $\quad R(P, P) \Rightarrow (\forall\langle P \longrightarrow_K Q\rangle : \exists\langle P \longrightarrow_{K'} Q\rangle, R(Q, Q))$.
Now the equality $I_K = I_{K'}$, the reachability condition of ppKSs (see Definition 9), and condition (*) imply that $\mathcal{PP}_K \subseteq \mathcal{PP}_{K'}$ and $\longrightarrow_K \subseteq \longrightarrow_{K'}$.
To prove the converse implication,

($\Leftarrow$) $\quad (\mathcal{PP}_K \subseteq \mathcal{PP}_{K'}) \wedge (\longrightarrow_K \subseteq \longrightarrow_{K'}) \Rightarrow K \preceq_{\text{sim}} K'$,
note that if K is a substructure of K', then K' simulates K via the identity relation $id \subseteq \mathcal{PP}_K \times \mathcal{PP}_{K'}$.

Notation. Since the two relations $\preceq_{\text{sim}}$ and $\preceq_{\text{sub}}$ on ppKSs are the same, we drop the subscripts and write $K \preceq K'$ for both relations.

Browne *et al* in [6] defined a notion of equivalence between two finite Kripke structures - usually called *bisimulation* in the literature [7]. Two states s and s' are called *bisimilar* if their labels of atomic propositions are the same and for any transition $s \longrightarrow t$ ($s' \longrightarrow t'$, respectively) there is a transition $s' \longrightarrow t'$ ($s \longrightarrow t$, respectively) such that t and t' are bisimilar. We say that two Kripke structures K, K' are *bisimilar* and write $K \approx K'$ if their initial states are bisimilar. Theorem 2 implies

Corollary 2 Given two ppKSs K and K' as above, the following three statements are equivalent:
(a) $\quad K \approx K'$
(b) $\quad K = K'$
(c) $\quad K \preceq K'$ and $K' \preceq K$

Proof: Since states in ppKSs are identified by their sets of labels, (a) $\Leftrightarrow$ (b) holds. (a) $\Leftrightarrow$ (c) is obvious. □

Thus, while for general Kripke structures *simulation equivalence* is weaker than bisimulation [7], this is not the case for ppKSs.

5 Modal Logic Theory of Feature Models

There is a rich structure in $\mathbb{P}(M)$ (for a given model M) that is not captured by the fact that $\mathbb{P}(M)$ is a ppKS—the class $\mathbf{KS}(F)$ is too big. We want to characterize $\mathbb{P}(M)$ in a more precise way by defining an as small as possible class of ppKSs to which $\mathbb{P}(M)$ would provably belong. Hence, we need a logic for defining classes of ppKSs by specifying a ppKS's properties.

In this section, we first introduce a modal logic called *partial product CTL* (ppCTL), which is tailored for specifying partial product Kripke structures' properties. Then, given a model M over a finite set of features F, we build two ppCTL theories from M's data, $\mathsf{ML}_{\subseteq}(M)$ and $\mathsf{ML}(M)$ (ML refers to Modal Logic), such that the former theory is a subset of the latter, and the following holds for any ppKS $K \in \mathbf{KS}(F)$:

Theorem 3 (Soundness) $\mathbb{P}(M) \models \mathsf{ML}(M)$.

Theorem 4 (Semi-completeness) $K \models \mathsf{ML}_{\subseteq}(M)$ implies $K \preceq \mathbb{P}(M)$.

Theorem 5 (Completeness) $K \models \mathsf{ML}(M)$ iff $K = \mathbb{P}(M)$.

Completeness allows us to replace models by the respective ppCTL-theories, which are highly amenable to formal analysis and automated processing. *Semi-completeness* is useful (as an auxiliary intermediate step to completeness, but also) for some important practical problems in feature modeling such as *refactoring* and *specialization* [38] and some other analysis operations [4] over models. (See Sect. 6 and Sect. 7.1 for more discussion.) We build theories $\mathsf{ML}_{\subseteq}(M)$ and $\mathsf{ML}(M)$ from small *component* theories, which specify the respective properties of M's PPL in terms of ppCTL.

The structure of the rest of the section is as follows: Sect. 5.1 introduces ppCTL. In Sect. 5.2, we start by discussing the structure of the entire component family, and explain how the compound theories, $\mathsf{ML}_{\subseteq}(M)$, $\mathsf{ML}(M)$, and $\mathsf{ML}_{+}(M) \overset{\text{def}}{=} \mathsf{ML}(M) \setminus \mathsf{ML}_{\subseteq}(M)$ are built from them. Then, in Section 5.3, we zoom into component theories and explain how they are built. Finally, in Sect. 5.4, we prove the correctness of the theorems. Sect. 5.5 discusses PPLs without $\mathsf{i2c}$. Appendix A.2 presents full the ppCTL theory of the model in Fig. 1.

5.1 Partial Product CTL (ppCTL)

Logic ppCTL is a fragment of CTL enriched with a constant (zero-ary) modality ! to capture full products.

Definition 11 (partial product CTL) Partial product CTL (ppCTL) formulas are defined using a finite set of propositional letters F, an ordinary signature of propositional connectives: constant (zero-ary) $\top$ (truth), unary $\neg$ (negation) and binary $\vee$ (disjunction) connectives, and a modal signature consisting of modal operators: constant (zero-ary) modality !, and three CTL unary modalities AX, AF, and AG. The *well-formed ppCTL-formulas* ϕ are given by the following grammar:

$$\phi ::= \ f \mid \top \mid \neg\phi \mid \phi \vee \phi \mid \mathsf{AX}\phi \mid \mathsf{AF}\phi \mid \mathsf{AG}\phi \mid ! \text{ , where } f \in F.$$

Other propositional and modal connectives are defined dually via negation as usual: $\bot$, $\wedge$, EX, EF, EG are the duals of $\top$, $\vee$, AX, AG, AF, respectively. Also, we define a unary modality $\Box^!\phi$ as a shorthand for $\mathsf{AG}(! \to \phi)$. Let *ppCTL*(F) denote the set of all ppCTL-formulas over F. □

The semantics of ppCTL-formulas is given using the class $\mathbf{KS}(F)$ of ppKSs built over the same set of features F. Let $K \in \mathbf{KS}(F)$ be a ppKS $(\mathcal{PP}, \longrightarrow, I)$. We first define a satisfaction relation $\models$ between a product $P \in \mathcal{PP}$ and a formula $\phi \in ppCTL(F)$ by structural induction on ϕ. This is done in Table 2.

Table 2: Rules of satisfiability

$P \models f$	iff $f \in P$ (for $f \in F$)
$P \models \top$	always holds
$P \models \neg\phi$	iff $P \not\models \phi$
$P \models \phi \vee \psi$	iff $(P \models \phi)$ or $(P \models \psi)$
$P \models \mathsf{AX}\phi$	iff $\forall\langle P \longrightarrow P'\rangle.\ P' \models \phi$
$P \models \mathsf{AF}\phi$	iff $\forall\langle P{=}P_1 \longrightarrow P_2 \longrightarrow \ldots\rangle\ \exists i \geq 1\colon\ P_i \models \phi$
$P \models \mathsf{AG}\phi$	iff $\forall\langle P{=}P_1 \longrightarrow P_2 \longrightarrow \ldots\rangle\ \forall i \geq 1\colon\ P_i \models \phi$
$P \models !$	iff $P \longrightarrow P$

Given two theories $\phi, \phi' \in ppCTL(F)$, we say that ϕ satisfies ϕ' and write $\phi \models \phi'$ iff $\forall K \in \mathbf{KS}(F) : K \models \phi \Rightarrow K \models \phi'$. We say that ϕ and ϕ' are *semantically equivalent* iff $\phi \models \phi'$ and $\phi' \models \phi$.

5.2 Structure of the component family

Table 3: Component and Compound Theories

M	Semi-completeness		To Ensure Completeness	Completeness
	BL	ML		
T	$\mathsf{BL}(T)$	$\varnothing$	$\mathsf{ML}^{\downarrow}_{+}(T)$	$\mathsf{ML}(T)$
$\mathcal{EX}$	$\mathsf{BL}(\mathcal{EX})$	$\varnothing$	$\varnothing$	$\mathsf{ML}(\mathcal{EX})$
$\mathcal{OR}$	$\varnothing$	$\mathsf{ML}^{!}_{\subseteq}(\mathcal{OR})$	$\varnothing$	$\mathsf{ML}^{!}(\mathcal{OR})$
$\mathcal{IN}$	$\varnothing$	$\mathsf{ML}^{!}_{\subseteq}(\mathcal{IN})$	$\varnothing$	$\mathsf{ML}^{!}(\mathcal{IN})$
i2c	$\mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}})$	$\mathsf{ML}^{\mathsf{i2c}\nrightarrow}_{\subseteq}(T_{\mathcal{OR}})$	$\varnothing$	$\mathsf{ML}^{\mathsf{i2c}}(T_{\mathcal{OR}})$
$\mathcal{FP}_M$	$\varnothing$	$\mathsf{ML}^{!}_{\subseteq}(M)$	$\mathsf{ML}^{!}_{+}(M)$	$\mathsf{ML}^{!}(M)$
$\mathcal{PP}_M$	$\mathsf{BL}(M)$	$\varnothing$	$\mathsf{ML}^{\downarrow}_{+}(T)\cup$ $\mathsf{ML}^{\leftrightarrow}_{+}(T_{\mathcal{OR}}, \mathcal{EX})$	$\mathsf{ML}^{\circ}(M)$
$\mathbb{P}(M)$	$\mathsf{ML}_{\subseteq}(M)$		$\mathsf{ML}_{+}(M)$	$\mathsf{ML}(M)$

All component theories we need are referenced in Table 3. Its bottom row consists of the three compound theories mentioned above; the last (rightmost) column theory is the union of the theories in its row—this is a general rule for the entire table. Another general rule is that each theory in the bottom row is the union of all components above it in its column(s) (and $\mathsf{ML}_{\subseteq}(M)$ is the union of all components in two columns). For further references, we call theories in the bottom row and the last column *external*; all other theories are *internal*.

Rows of the table are indexed by structural *concerns* to be logically encoded; columns are named by the goals of these encodings: to provide semi-completeness with respect to full product line and PPL (split into Boolean and modal components), and to provide completeness with respect to full product line and PPL: a theory in the last column is the union of all theories in its row, and thus ensures completeness with respect to the concern corresponding to the row. A theory in this column is called the *complete* theory of the corresponding component. Each internal theory is an encoding of the corresponding concern for the corresponding goal. For example, theory $\mathsf{ML}^{!}_{\subseteq}(\mathcal{OR})$ modally specifies the $\mathcal{OR}$ structure to provide semi-completeness with respect to full product line (note the ! superindex). For another example, $\mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}})$ is a Boolean encoding of the $\mathsf{i2c}$-principle, and its neighbor on the right is the additional modal constraint for the same concern—it is needed to ensure semi-completeness. The empty neighbour on the right means that nothing should be added (for this concern) to ensure completeness. We do not intend to make the table strictly logical: its goal is to reference component theories and explain their intentions.

5.3 The Content of Component Theories

Now we specify the internal theories, and explain their meaning. Boolean theories are specified in Table 1. Modal theories are defined in Table 4 based on the following motivation.

Table 4: Definitions of (basic) ppCTL theories

$$\mathsf{ML}^{\downarrow}_{+}(T) = \big\{ f \wedge \neg \bigvee f_{\downarrow} \wedge \neg \bigvee \mathsf{BL}^{\mathcal{EX}}(g) \rightarrow \mathsf{EX}g : f \in F, f_{\downarrow} \neq \varnothing, g \in f_{\downarrow} \big\}$$

$$\mathsf{ML}^{!}_{\subseteq}(\mathcal{OR}) = \{ f \rightarrow \Box^{!} \bigvee G : f \in F, G \in \mathcal{OR}(f) \}$$

$$\mathsf{ML}^{!}_{\subseteq}(\mathcal{IN}) = \{ \bigwedge G \rightarrow \Box^{!} \bigvee G' : (G, G') \in \mathcal{IN} \}$$

$$\mathsf{ML}^{!}_{\subseteq}(M) = \{ ! \rightarrow \ \bigwedge \mathsf{BL}^{!}(M) \}$$

$$\mathsf{ML}^{!}_{+}(M) = \{ \bigwedge \mathsf{BL}^{!}(M) \rightarrow ! \}$$

$$\mathsf{ML}^{\mathsf{i2c}\text{-}\nrightarrow}_{\subseteq}(T_{\mathcal{OR}}) = \big\{ f \wedge \neg g \wedge \neg \bigwedge \mathsf{BL}^{!}(T^{f}_{\mathcal{OR}}) \rightarrow \neg \mathsf{EX}g : f, g \in F, f \neq g, f^{\uparrow} = g^{\uparrow} \big\}$$

$$\mathsf{ML}^{\leftrightarrow}_{+}(T_{\mathcal{OR}}, \mathcal{EX}) = \big\{ f^{\uparrow} \wedge \bigwedge \mathsf{BL}^{\mathsf{i2c}}(f) \wedge \neg \bigvee \mathsf{BL}^{\mathcal{EX}}(f) \rightarrow \mathsf{EX}f : f \in F \big\},$$

$$\mathsf{BL}^{\mathsf{i2c}}(f) = \{ g \rightarrow \bigwedge \mathsf{BL}^{!}(T^{g}_{\mathcal{OR}}) : g, f \in F, g^{\uparrow} = f^{\uparrow}, g \neq f \}$$

$$\mathsf{BL}^{\mathcal{EX}}(f) = \big\{ \bigwedge (G \setminus \{f\}) : G \in \mathcal{EX}, f \in G \big\}$$

The theory $\mathsf{ML}^{\downarrow}_{+}(T)$ states that if a feature f is visited in a current state (partial product) without visiting any of its children (note $\neg \bigvee f_{\downarrow}$ in the theory), then, for each child g of f, if adding g to the current state does not violate the exclusive constraints (note $\neg \bigvee \mathsf{BL}^{\mathcal{EX}}(g)$ in the theory), then there must be a state immediately accessible from the current state visiting g, i.e., $f \wedge \neg \bigvee f_{\downarrow} \wedge \neg \bigvee \mathsf{BL}^{\mathcal{EX}}(g) \rightarrow \mathsf{EX}g$. The union of this theory and $\mathsf{BL}(T)$ generates a complete theory $\mathsf{ML}(T)$ (Table 3). A ppKS K satisfying $\mathsf{ML}(T)$ is guaranteed to capture the tree structure T.

Since exclusive constraints in a model talk only about semi-completeness of partial products, the corresponding $_{\mathsf{ML}+}$ theory is empty. Thus, $\mathsf{ML}(\mathcal{EX}) = \mathsf{BL}(\mathcal{EX})$.

The theories corresponding to $\mathcal{OR}$ deal with full products (states with self-loop transitions). The theory $\mathsf{ML}^!_{\subseteq}(\mathcal{OR})$ is the modal version of the Boolean theory $\mathsf{BL}^!(\mathcal{OR})$ (Table 1). Consider an OR group G with $G^{\uparrow} = f$. The theory $\mathsf{ML}^!_{\subseteq}(\mathcal{OR})$ states that if f is visited in a current state, then at least one of the elements involved in G must be visited in any final products accessible from the current state, i.e., $f \rightarrow \Box^! \bigvee G$.

The nature of the theory corresponding to $\mathcal{IN}$ is like $\mathcal{OR}$'s: it also deals only with full products. The theory $\mathsf{ML}^!_{\subseteq}(\mathcal{IN})$ is the modal version of the Boolean theory $\mathsf{BL}^!(\mathcal{IN})$. Let (G, G') be an inclusive constraint. The theory $\mathsf{ML}^!_{\subseteq}(\mathcal{IN})$ states that if all the elements involved in G are visited in a current state, then at least one of the elements in G' must be visited in any final products accessible from the current state, i.e., $\bigwedge G \rightarrow \Box^! \bigvee G'$.

Obviously, the two theories $\mathsf{ML}^!_{\subseteq}(\mathcal{OR})$ and $\mathsf{ML}^!_{\subseteq}(\mathcal{IN})$ are derivable from the theory $\mathsf{ML}^!_{\subseteq}(M)$. $\mathsf{ML}^!_{\subseteq}(M)$ holding in a ppKS guarantees that any full product in the ppKS is a full product of M. On the other hand, any ppKS satisfying the theory $\mathsf{ML}^!(M)$ $(= \mathsf{ML}^!_{\subseteq}(M) \cup \mathsf{ML}^!_{+}(M))$ must include all full products of M and only them.

Recall that the theory $\mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}})$ (Table 1) guarantees that the partial products of the PPL respect the $\mathsf{i2c}$ principle. However, as discussed in Sect. 2.3, transitions also have to respect this principle. The modal theory $\mathsf{ML}^{\mathsf{i2c}\nrightarrow}_{\subseteq}(T_{\mathcal{OR}})$ excludes the invalid transitions due to the $\mathsf{i2c}$ principle (see Table 4). This theory states that if a feature f is visited in a current state without being completely instantiated (note $\neg \bigwedge \mathsf{BL}^!(T^f_{\mathcal{OR}})$ in the theory), then there must not be any states immediately accessible from the current state including any newly added sibling g of f, i.e., for any sibling g of f: $f \wedge \neg g \wedge \neg \bigwedge \mathsf{BL}^!(T^f_{\mathcal{OR}}) \rightarrow \neg \mathsf{EX}g$. Then, the complete theory relating to $\mathsf{i2c}$, $\mathsf{ML}^{\mathsf{i2c}}(T_{\mathcal{OR}})$, would be the union of $\mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}})$ and $\mathsf{ML}^{\mathsf{i2c}\nrightarrow}_{\subseteq}(T_{\mathcal{OR}})$.

Recall that, according to Definition 5, a set of features is a valid partial product iff it satisfies the Boolean theory $\mathsf{BL}(M)$. However, any ppKS satisfying this theory does not necessarily include all valid partial products. To ensure that the ppKS includes all partial products, we add modal theories $\mathsf{ML}^{\downarrow}_{+}(T)$ and $\mathsf{ML}^{\leftrightarrow}_{+}(T_{\mathcal{OR}}, \mathcal{EX})$. Consider a state P and a feature f such that $f \notin P$ and $f^{\uparrow} \in P$. The theory $\mathsf{ML}^{\leftrightarrow}_{+}(T_{\mathcal{OR}}, \mathcal{EX})$ states that if adding f to P does not violate the exclusive constraints and the $\mathsf{i2c}$ principle (note $\bigwedge \mathsf{BL}^{\mathsf{i2c}}(f)$ and $\neg \bigvee \mathsf{BL}^{\mathcal{EX}}(f)$ in the theory, respectively), then there must be an immediately accessible state from P including f, i.e., $f^{\uparrow} \wedge \bigwedge \mathsf{BL}^{\mathsf{i2c}}(f) \wedge \neg \bigvee \mathsf{BL}^{\mathcal{EX}}(f) \rightarrow \mathsf{EX}f$. The corresponding complete theory is denoted by $\mathsf{ML}_{+}(M)$ and is equal to $\mathsf{BL}(M) \cup \mathsf{ML}^{\downarrow}_{+}(T) \cup \mathsf{ML}^{\leftrightarrow}_{+}(T_{\mathcal{OR}}, \mathcal{EX})$.

Any ppKS K satisfying the semi-completeness theory $\mathsf{ML}_{\subseteq}(M)$ would be a substructure of $\mathbb{P}(M)$, i.e., $\mathbb{P}(M)$ simulates K. On the other hand, the theory $\mathsf{ML}(M)$, which is the union of $\mathsf{ML}_{\subseteq}(M)$ and $\mathsf{ML}_{+}(M)$, guarantees completeness, i.e., any ppKS K satisfying $\mathsf{ML}(M)$ is equal to the PPL of M. These are proven in the next subsection.

5.4 Soundness, Semi-Completeness, and Completeness: Proofs

Our plan is as follows. We first prove soundness, then semi-completeness. The completeness theorem will be a direct corollary of Lemma 1 and Lemma 2.

Soundness: $\mathbb{P}(M) \models \mathsf{ML}(M)$.
Proof: To prove this theorem, we need to show that $\mathbb{P}(M)$ satisfies any components of the theory $\mathsf{ML}(M)$.

(a) $\mathbb{P}(M) \models \mathsf{BL}(M)$ is obvious by to Definition 5. Thus, all the Boolean theories from Table 3 are satisfied by $\mathbb{P}(M)$.

(b) $\mathbb{P}(M) \models \mathsf{ML}^{\downarrow}_{+}(T)$:

Let $P \in \mathcal{PP}_M$, $f \in P$, $g \in f_{\downarrow}$, $P \cap f_{\downarrow} = \varnothing$, and $P \not\models \bigvee \mathsf{BL}^{\mathcal{EX}}(g)$, i.e., $P \models f \wedge \neg \bigvee f_{\downarrow} \wedge \neg \bigvee \mathsf{BL}^{\mathcal{EX}}(g)$. We want to show that $P \models \mathsf{EX}g$. Let $P' = P \cup \{g\}$. According to (a), $P \models \mathsf{BL}(T) \cup \mathsf{BL}(\mathcal{EX})$. Since the g's parent is already in P', adding g to P does not violate $\mathsf{BL}(T)$. Since $P \not\models \bigvee \mathsf{BL}^{\mathcal{EX}}(g)$, adding g to P also does not violate $\mathsf{BL}(\mathcal{EX})$. Therefore, $P' \models \mathsf{BL}(T) \cup \mathsf{BL}(\mathcal{EX})$. Since all subfeatures of f are absent in P, $\mathsf{BL}^{\mathsf{i2c}}(P, f) = \varnothing$ (note Definition 6) and hence $P \models \mathsf{BL}^{\mathsf{i2c}}(P, f)$. Since $P' \models \mathsf{BL}(T) \cup \mathsf{BL}(\mathcal{EX})$ and $P \models \mathsf{BL}^{\mathsf{i2c}}(P, f)$, according to Definition 7, there is a transition $P \longrightarrow_M P'$. Therefore, $P \models \mathsf{EX}g$.

(c) $\mathbb{P}(M) \models \mathsf{ML}^{!}(M)$ follows obviously, since the set of states with self-loops in $\mathbb{P}(M)$ is equal to the set of all full products of M. Note that this also implies that $\mathbb{P}(M)$ satisfies both theories $\mathsf{ML}^{!}_{\subseteq}(\mathcal{OR})$ and $\mathsf{ML}^{!}_{\subseteq}(\mathcal{IN})$, since these two theories are derivable from the theory $\mathsf{ML}^{!}_{\subseteq}(M)$.

(d) $\mathbb{P}(M) \models \mathsf{ML}^{\mathsf{i2c}\text{-}\not\to}_{\subseteq}(T_{\mathcal{OR}})$ follows obviously. Indeed, this theory guarantees that there would not be an invalid transition due to $\mathsf{i2c}$ principle.

(e) $\mathbb{P}(M) \models \mathsf{ML}^{\leftrightarrow}_{+}(T_{\mathcal{OR}}, \mathcal{EX})$:

Let f and P be a feature and a partial product of M, respectively, such that $f^{\uparrow} \in P$, $P \models \mathsf{BL}^{\mathsf{i2c}}(f)$, and $P \not\models \bigvee \mathsf{BL}^{\mathcal{EX}}(f)$. There are two cases: (1) $f \in P$, (2) $f \notin P$. Due to the singletonicity condition of ppKSs (Definition 9), (1) trivially leads us to the result. In case (2): According to Definition 7, there exists a transition $P \longrightarrow_M P \cup \{f\}$, which implies $P \models \mathsf{EX}\ f$. This results in $\mathbb{P}(M) \models \mathsf{ML}^{\leftrightarrow}_{+}(T_{\mathcal{OR}}, \mathcal{EX})$.

Note that any other theory is the union of some of the above theories. The theorem is proven. □

Semi-completeness: $K \models \mathsf{ML}_{\subseteq}(M)$ implies $K \preceq \mathbb{P}(M)$.

Proof: Let $K \models \mathsf{ML}_{\subseteq}(M)$. Since $K \models \mathsf{BL}(M)$, according to Definition 5, $\mathcal{PP}_K \subseteq \mathcal{PP}_M$. Now, we are going to show that $\longrightarrow_K \subseteq \longrightarrow_M$.

Due to $K \models \mathsf{ML}^{!}_{\subseteq}(M)$ and $\mathcal{PP}_K \subseteq \mathcal{PP}_M$, any self-loop transitions $P \longrightarrow_K P$ in K is a self-loop transition $P \longrightarrow_M P$ in $\mathbb{P}(M)$.

Consider a transition $P \longrightarrow_K P'$, where $P' = P \cup \{f\}$ for a feature $f \notin P$. We want to show that there is a transition $P \longrightarrow_M P'$ in $\mathbb{P}(M)$. Again, note that any state in K is a partial product of M. To prove this statement, according to Definition 7, we need to show that (a1) $P' \models \mathsf{BL}(T)$, (a2) $P' \models \mathsf{BL}(\mathcal{EX})$, and (a3) $P \models \mathsf{BL}^{\mathsf{i2c}}(P, f)$. (a1) and (a2) are immediate corollaries of $K \models \mathsf{BL}(M)$. To prove (a3), we need to show that for any siblings g with $g \in P$, $P \models \mathsf{BL}^{!}(T^{g}_{\mathcal{OR}})$ (see Definition 6). Assume by a way of contradiction that $P \not\models \mathsf{BL}^{!}(T^{g}_{\mathcal{OR}})$, i.e., g is not completely instantiated in P. Since $K \models \mathsf{ML}^{\mathsf{i2c}\text{-}\not\to}_{\subseteq}(T_{\mathcal{OR}})$, $g \in P$, and $P \not\models \mathsf{BL}^{!}(T^{g}_{\mathcal{OR}})$, there must not be a transition $P \longrightarrow_K P'$. This leads us to a contradiction. Thus, (a3) holds. Based on the above reasonings, $\longrightarrow_K \subseteq \longrightarrow_M$.

Since $\mathcal{PP}_K \subseteq \mathcal{PP}_M$ and $\longrightarrow_K \subseteq \longrightarrow_M$, according to Theorem 2, $K \preceq \mathbb{P}(M)$. □

Completeness: $K \models \mathsf{ML}(M)$ iff $K = \mathbb{P}(M)$.

To prove the completeness theorem, we first need the following lemmas 1 and 2.

Lemma 1 $K \models \mathsf{ML}^{\circ}(M)$ implies $\mathcal{PP}_K = \mathcal{PP}_M$. □

Proof: Let $K \models \mathsf{ML}^{\circ}(M)$. By Theorem 4, $\mathcal{PP}_K \subseteq \mathcal{PP}_M$. Now we need to show that $\mathcal{PP}_M \subseteq \mathcal{PP}_K$. (We will illustrate general constructs used in the proof with our running example - follow the footnotes.)

Let $P \in \mathcal{PP}_M$ and r be the root feature of T. The features included in P represent a subtree of T, denoted by T_P, whose root is r.[10]

We do a pre-order depth-first traversal of T_P of a special kind complying to $\mathsf{i2c}$-principle: in each level of the tree, all the nodes that are completely instantiated must be visited before the other nodes.[11]

Let $S_P = \langle f_1, \dots, f_n \rangle$ with $f_1 = r$ be the traversal of T_P.

The following condition (R) holds:[12]

(R): for all $i < n$ either

(R-1) $f_i = f_{i+1}^{\uparrow}$ or

(R-2) $\exists \langle j < i \rangle : f_j = f_{i+1}^{\uparrow}$ & $\forall g \in \{f_1, \dots, f_i\} : \big(g^{\uparrow} = f_{i+1}^{\uparrow}\big) \Rightarrow \big(\{f_1, \dots, f_i\} \models \mathsf{BL}^{!}(T^{g}_{\mathcal{OR}})\big)$, i.e., g is completely instantiated in $\{f_1, \dots, f_i\}$.

We prove that any prefix subsequence of S_P is a partial product of K and so P itself. To this end, we use the following inductive reasoning:

(*base case*): $K \models r$ implies that $I_K = \{r\} = \{f_1\}$.

(*hypothesis*): Assume that, for some $1 \le i < n$, any prefix of the sequence $\langle f_1, \dots, f_i \rangle$ is a state in K and there exists a path $\{f_1\} \longrightarrow_K \cdots \longrightarrow_K \{f_1, \dots, f_i\}$. Let $P' = \{f_1, \dots, f_i\}$.

(*inductive step*): We want to prove that any prefix of the sequence $\langle f_1, \dots f_i, f_{i+1} \rangle$ is a state in K and there exists the path $\{f_1\} \longrightarrow_K \cdots \longrightarrow_K P' \longrightarrow_K P' \cup \{f_{i+1}\}$. To this end, we need to show that $P' \cup \{f_{i+1}\} \in \mathcal{PP}_K$ and there exists a transition $P' \longrightarrow_K P' \cup \{f_{i+1}\}$. We will prove this for both cases (R-1) and (R-2) introduced above:

(R-1). Since $P \models \mathsf{BL}(\mathcal{EX})$ (note that $P \in \mathcal{PP}_M$), adding f_{i+1} to P' does not violate the exclusive constraints, i.e., $P' \not\models \bigvee \mathsf{BL}^{\mathcal{EX}}(f_{i+1})$. As f_i is freshly added to state P', $P' \not\models \bigvee f_{i\downarrow}$. Therefore, $P' \models f_i \wedge \neg \bigvee f_{i\downarrow} \wedge \neg \bigvee \mathsf{BL}^{\mathcal{EX}}(f_{i+1})$. Due to $K \models \mathsf{ML}^{\downarrow}_{+}(T)$, this implies that there is a transition $P' \longrightarrow_K P' \cup \{f_{i+1}\}$. Hence, $\{f_1, \dots, f_{i+1}\} \in \mathcal{PP}_K$.

(R-2). As $\forall g \in P' : (g^{\uparrow} = f_{i+1}^{\uparrow}) \Rightarrow (P' \models \mathsf{BL}^{!}(T^{g}_{\mathcal{OR}}))$ (note (R-2) above), adding f_{i+1} to P' does not violated $\mathsf{i2c}$,i.e., $P' \models \mathsf{BL}^{\mathsf{i2c}}(f_{i+1})$.

[10]For an example, consider the partial product $\{\mathsf{car}, \mathsf{eng}, \mathsf{gear}, \mathsf{mnl}, \mathsf{oil}\}$ in the model in Fig. 1. We have the following formulas corresponding to $\mathsf{BL}(T)$: $\mathsf{eng} \rightarrow \mathsf{car}$, $\mathsf{gear} \rightarrow \mathsf{car}$, $\mathsf{mnl} \rightarrow \mathsf{gear}$, and $\mathsf{oil} \rightarrow \mathsf{gear}$, which clearly represent the subtree $(\mathsf{eng}) \rightarrow \mathsf{car} \leftarrow (\mathsf{mnl} \rightarrow \mathsf{gear} \leftarrow \mathsf{oil})$.

[11]In the running example, gear must be visited before eng, since it is completely disassembled in $\{\mathsf{car}, \mathsf{eng}, \mathsf{gear}, \mathsf{mnl}, \mathsf{oil}\}$. In this example, the traversal would result in the sequence $\langle \mathsf{car}, \mathsf{gear}, \mathsf{mnl}, \mathsf{oil}, \mathsf{eng} \rangle$.

[12]$\mathsf{car} = \mathsf{gear}^{\uparrow}$; $\mathsf{gear} = \mathsf{mnl}^{\uparrow}$; gear (resp. car) $= \mathsf{oil}^{\uparrow}$ (resp. $\mathsf{eng}^{\uparrow}$) and mnl (resp. gear) is the only sibling of oil (resp. eng) which is completely instantiated.

$P \models \mathsf{BL}(\mathcal{EX})$ implies that any subset of P satisfies $\mathsf{BL}(\mathcal{EX})$. Since $P' \cup \{f_{i+1}\} \subseteq P$, $P' \cup \{f_{i+1}\} \models \mathsf{BL}(\mathcal{EX})$, which means $P' \not\models \bigvee \mathsf{BL}^{\mathcal{EX}}(f_{i+1})$.

Since $P' \models \mathsf{BL}^{\mathsf{i2c}}(f_{i+1}) \wedge \neg \bigvee \mathsf{BL}^{\mathcal{EX}}(f_{i+1}) \wedge f_{i+1}^{\uparrow}$, and $K \models \mathsf{ML}_{+}^{\leftrightarrow}(T_{\mathcal{OR}}, \mathcal{EX})$, there is a state $\{f_1, \ldots, f_{i+1}\} \in \mathcal{PP}_K$ such that $P' \longrightarrow_K P' \cup \{f_{i+1}\}$. Hence, $P \in \mathcal{PP}_K$. □

Lemma 2 $K \models \mathsf{ML}(M)$ implies $\longrightarrow_K = \longrightarrow_M$. □

Proof: Let $K \models \mathsf{ML}(M)$. There are two types of transitions in a ppKS: self-loop transitions and others. Note that self-loop transitions denote full products. We show that (1) full products of both $\mathbb{P}(M)$ and K are the same, i.e., the set of their self-loops are the same, (2) Non-loop transitions in K and $\mathbb{P}(M)$ are the same. (1) is obvious, since $K \models \mathsf{ML}^{!}(M)$ (note Table 1). In the following we also show that the statement (2) holds.

According to Theorem 4, $\longrightarrow_K \subseteq \longrightarrow_M$. Now what we need is to prove that any non-loop transition in $\mathbb{P}(M)$ is also a transition in K. Note that, according to Lemma 1, $\mathcal{PP}_K = \mathcal{PP}_M$. Consider a transition $P \longrightarrow_M P'$, where $P' = P \cup \{f\}$ for a feature $f \notin P$. We want to show that there is a transition $P \longrightarrow_K P'$ in K. According to Definition 7, $P' \models \mathsf{BL}(T) \cup \mathsf{BL}(\mathcal{EX})$, and $P \models \mathsf{BL}^{\mathsf{i2c}}(P, f)$. Thus, there are two choices:

(i) $\mathsf{BL}^{\mathsf{i2c}}(P, f) = \varnothing$

(ii) $\mathsf{BL}^{\mathsf{i2c}}(P, f) \neq \varnothing$

(i): This implies that the parent of f is freshly added through a transition ingoing to P. Hence, due to $K \models \mathsf{ML}_{+}^{\downarrow}(T)$, there exists a transition $P \longrightarrow_K P'$.

(ii): Since $P' \models \mathsf{BL}(\mathcal{EX})$, $P \models \neg \bigvee \mathsf{BL}^{\mathcal{EX}}(f)$. Also, $P \models \mathsf{BL}^{\mathsf{i2c}}(P, f)$ implies that $P \models \mathsf{BL}^{!}(T_{\mathcal{OR}}^{g})$ for any $g \in P \cap (f^{\uparrow})_{\downarrow}$, which means $P \models \mathsf{BL}^{\mathsf{i2c}}(f)$. Hence, due to $\mathsf{ML}_{+}^{\leftrightarrow}(T_{\mathcal{OR}}, \mathcal{EX})$, there exists a transition $P \longrightarrow_K P'$.

(i) and (ii) implies that any non-loop transition in $\mathbb{P}(M)$ is also a transition in K. Hence, $\longrightarrow_M \subseteq \longrightarrow_K$. □

Proof of Theorem 5 (Completeness):
Lemma 1 shows that $K \models \mathsf{ML}^{\circ}(M)$ implies $\mathcal{PP}_K = \mathcal{PP}_M$. Lemma 2 proves that $K \models \mathsf{ML}(M)$ implies $\longrightarrow_K = \longrightarrow_M$. Hence, $K \models \mathsf{ML}(M)$ implies $K = \mathbb{P}(M)$. Considering the soundness theorem (Theorem 3), the completeness theorem is proven. □

5.5 Feature modeling without $\mathsf{i2c}$

The $\mathsf{i2c}$-principle should not be considered as a mandatory requirement in generating PPLs from models. Given a model M, let $\mathbb{P}^{-\mathsf{i2c}}(M)$ denote its PPL, whose partial products and transitions do not necessarily meet the $\mathsf{i2c}$ principle: its set of states is $\{P \subseteq F : P \models \mathsf{BL}(M) \setminus \mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}})\}$, denoted by $\mathcal{PP}_M^{-\mathsf{i2c}}$, and its transitions, denoted by $\longrightarrow_M^{-\mathsf{i2c}}$, are defined as in Definition 7, but the condition (a3) is not required. To build the ppCTL theory of $\mathbb{P}^{-\mathsf{i2c}}(M)$, we just need to subtract the $\mathsf{i2c}$ theories from $\mathsf{ML}(M)$ (i.e., exclude the column $\mathsf{i2c}$ from Table 3). Let $\mathsf{ML}^{-\mathsf{i2c}}(M)$ and $\mathsf{ML}_{\subseteq}^{-\mathsf{i2c}}(M)$ denote the corresponding semi-complete and complete theories for $\mathbb{P}^{-\mathsf{i2c}}(M)$, respectively. Then our proofs above provide also the following results (just ignore the parts corresponding to $\mathsf{i2c}$).

Theorem 6 $\mathbb{P}^{-\mathsf{i2c}}(M) \models \mathsf{ML}^{-\mathsf{i2c}}(M)$. □

Theorem 7 $K \models \mathsf{ML}_{\subseteq}^{-\mathsf{i2c}}(M)$ implies $K \preceq \mathbb{P}^{-\mathsf{i2c}}(M)$. □

Theorem 8 $K \models \mathsf{ML}^{-\mathsf{i2c}}(M)$ iff $K = \mathbb{P}^{-\mathsf{i2c}}(M)$. □

Product lines without the $\mathsf{i2c}$-principle have several interesting properties.

Definition 12 We call a ppKS K *Boolean*, if a transition between two distinct states $P, P' \in \mathcal{PP}_K$ exists iff $P \subset P'$ and $P' \setminus P$ is a singleton. The class of all Boolean ppKSs over a set of features F is denoted by $\mathbf{KS}^{\mathsf{BL}}(F)$; thus, $\mathbf{KS}^{\mathsf{BL}}(F) \subset \mathbf{KS}(F)$. □

Note that any Boolean ppKS K is determined by a pair of Boolean theories $(\Psi_K, \Psi_K^!)$ such that $\Psi_K^! \models \Psi_K$: the first theory defines all products in K, while the second theory defines (the subset of) full products.[13]

Now it is easy to see that for a given model M, its PPL $\mathbb{P}^{-\mathsf{i2c}}(M)$ is a Boolean ppKS specified by the pair $(\mathsf{BL}^{-\mathsf{i2c}}(M), \mathsf{BL}^!(M))$ with the first theory, $\mathsf{BL}^{-\mathsf{i2c}}(M) = \mathsf{BL}(M) \setminus \mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}})$ (see Table 1), specifying partial products not necessary satisfying the $\mathsf{i2c}$, and the second one specifying full products. This ppKS enjoys the following *universal maximality* property.

Proposition 2 Let $M \in \mathbf{FM}(F)$ be a model. Then any Boolean ppKS $K \in \mathbf{KS}^{\mathsf{BL}}(F)$ such that $K \models \mathsf{BL}^{-\mathsf{i2c}}(M)$ and $K \models (! \to \mathsf{BL}^!(M))$, is a substructure of $\mathbb{P}^{-\mathsf{i2c}}(M)$, i.e., $K \preceq \mathbb{P}^{-\mathsf{i2c}}(M)$.

[13]In detail, $\Psi_K = \bigvee\{\Psi(P)\colon\ P \in \mathcal{PP}_K\}$ and for a product $P = \{f_1, \ldots, f_n\}$, $\Psi(P) = \bigwedge\{f_1, \ldots, f_n, \neg g_1, \ldots \neg g_k\}$, where features $g_j \notin P$, and $\neg g_j$ denotes negation of g_j. Theory $\Psi_K^!$ is defined similarly with full rather than partial products.

Proof: Let K be a ppKS as above. $K \models \mathsf{BL}^{-\mathsf{i2c}}(M)$ implies that $\forall P \in \mathcal{PP}_K : P \models \mathsf{BL}^{-\mathsf{i2c}}(M)$. Therefore, any state in K is a partial product not necessary satisfying the $\mathsf{i2c}$ of M, i.e., $\mathcal{PP}_K \subseteq \mathcal{PP}_M^{-\mathsf{i2c}}$.
Since both K and $\mathbb{P}^{-\mathsf{i2c}}(M)$ are in $\mathbf{KS}^{\mathsf{BL}}(F)$ and $\mathcal{PP}_K \subseteq \mathcal{PP}_M^{-\mathsf{i2c}}$, $\forall P, P' \in \mathcal{PP}_K : P \neq P' \wedge P \longrightarrow_K P' \Rightarrow P \longrightarrow_M^{-\mathsf{i2c}} P'$. This means that any non-loop transition in K is a non-loop transition in $\mathbb{P}^{-\mathsf{i2c}}(M)$. Now, we just need to show that any self-loop transition in K is a self-loop transition in $\mathbb{P}^{-\mathsf{i2c}}(M)$. Since $K \models\, ! \rightarrow \mathsf{BL}^{!}(M)$, any full product of K is a full product of M, i.e., $\forall P \in \mathcal{PP}_K : P \longrightarrow_K P \Rightarrow P \longrightarrow_M^{-\mathsf{i2c}} P$. $\square$

6 Feature model refactoring

Model refactoring is important for the practice of feature modeling [38]. The goal of refactoring is to replace a given model M with a syntactically different but semantically equivalent model M', i.e., having the same (partial) products. In this section we investigate what we can say about syntactical relationships of two semantically equivalent models over the same set of features F, i.e., such that $M, M' \in \mathbf{FM}(F)$ and $\mathbb{P}(M) = \mathbb{P}(M')$.

The first lemma shows that their trees must be identical.

Lemma 3 Given two models M and M', $\mathbb{P}(M) = \mathbb{P}(M') \Rightarrow T_M = T_{M'}$.

Proof: Let $T_M = (F, r, \_^{\uparrow})$ and $T_{M'} = (F, r, \_^{\uparrow'})$. Consider an arbitrary feature $f \in F$. We show that $f_{\downarrow\downarrow} \subseteq f_{\downarrow'\downarrow'}$ and $f_{\downarrow'\downarrow'} \subseteq f_{\downarrow\downarrow}$, i.e., $f_{\downarrow\downarrow} = f_{\downarrow'\downarrow'}$. Assume that there is a feature $g \in F$ such that $g \in f_{\downarrow\downarrow}$ but $g \notin f_{\downarrow'\downarrow'}$. The latter implies that $\exists P \in \mathcal{PP} : g \in P \wedge f \notin P$, whereas the former implies that $\forall P \in \mathcal{PP} : g \in P \Rightarrow f \in P$. As these two statements contradict each other, $f_{\downarrow\downarrow} \subseteq f_{\downarrow'\downarrow'}$. Likewise, $f_{\downarrow'\downarrow'} \subseteq f_{\downarrow\downarrow}$. Therefore, $f_{\downarrow\downarrow} = f_{\downarrow'\downarrow'}$ for any feature $f \in F$, which implies (together with equality between the root features) that $\_^{\uparrow} = \_^{\uparrow'}$. Thus, $T_M = T_{M'}$ $\square$

Given a set of Boolean formulas (a theory) Ψ, we denote its semantic closure $\{\phi\colon\ \Psi \models \{\phi\}\}$ by $\Psi^{\models}$ (see footnote 7 on page 81 for the definition). Given a model M, we mean $\mathsf{BL}(\mathcal{EX}_M)^{\models}$ by $\mathcal{EX}_M^{\models}$.

Lemma 4 Given two models M and M', $\mathbb{P}(M) = \mathbb{P}(M') \Rightarrow \mathcal{EX}_M^{\models} = \mathcal{EX}_{M'}^{\models}$, i.e., their sets of exclusive constraints are equivalent.[14]

[14]Note that an exclusive constraint in a model may be derivable from others, e.g., if $G \in \mathcal{EX}$, then for any feature $f \notin G$, $G \cup \{f\}$ is a derivable exclusive constraint. This is why we have used semantically equality rather than equality.

Proof: Let M and M' be two models such that $\mathbb{P}(M) = \mathbb{P}(M')$. According to Lemma 3, $T_M = T_{M'}$. Therefore, their corresponding set of incomparable features are the same, i.e., $(\#2^F)_M = (\#2^F)_{M'}$. Let us denote this set by $\#2^F$.

According to the definition of partial products,
$\forall G \in \#2^F : \wedge G \in \mathcal{EX}_M^{\models} \Leftrightarrow (\nexists P \in \mathcal{PP}_M : P \models \bigwedge G)$,
$\forall G \in \#2^F : \wedge G \in \mathcal{EX}_{M'}^{\models} \Leftrightarrow (\nexists P \in \mathcal{PP}_{M'} : P \models \bigwedge G)$.
Since $\mathcal{PP}_M = \mathcal{PP}_{M'}$, $\mathcal{EX}_M^{\models}$ and $\mathcal{EX}_{M'}^{\models}$ must be equal. In other words, $\mathcal{EX}_M$ and $\mathcal{EX}_{M'}$ are semantically equivalent. □

Given a model M, we write $(\mathcal{IN}_M \cup \mathcal{OR}_M)^{\models}$ to denote $(\mathsf{BL}(\mathcal{IN}_M) \cup \mathsf{BL}(\mathcal{IN}_M))^{\models}$.

Lemma 5 Given two models M and M', $\mathbb{P}(M) = \mathbb{P}(M') \Rightarrow (\mathcal{IN}_M \cup \mathcal{OR}_M)^{\models} = (\mathcal{IN}_{M'} \cup \mathcal{OR}_{M'})^{\models}$.

Proof: Let M and M' be two models with $\mathbb{P}(M) = \mathbb{P}(M')$. Then, their full products are the same (recall that full products are specified by self-loops), which means that $\mathsf{BL}^!(M)^{\models} = \mathsf{BL}^!(M')^{\models}$ (see row (all$^!$) in Table 1). According to Lemma 3 and Lemma 4, $(\mathsf{BL}(T_M) \cup \mathsf{BL}(\mathcal{EX}_M))^{\models} = (\mathsf{BL}(T_{M'}) \cup \mathsf{BL}(\mathcal{EX}_{M'}))^{\models}$. This implies that $(\mathsf{BL}^!(\mathcal{OR}_M) \cup \mathsf{BL}^!(\mathcal{IN}_M))^{\models} = (\mathsf{BL}^!(\mathcal{OR}_{M'}) \cup \mathsf{BL}^!(\mathcal{IN}_{M'}))^{\models}$. □

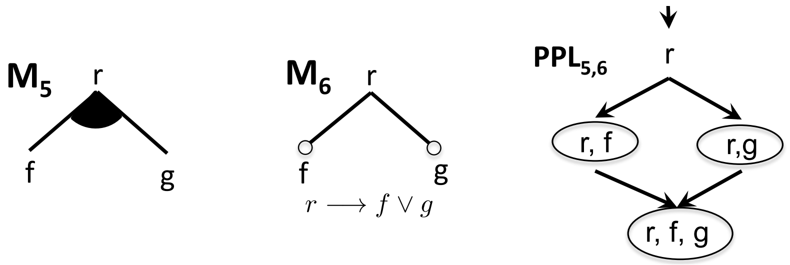


Figure 5: M_5 and M_6 are semantically equal.

Note that PPLs cannot distinguish between $\mathcal{OR}$ and $\mathcal{IN}$ constraints in models, as we can replace an OR constraint with an inclusive CC. For instance, consider the models $M_{5,6}$ in Fig. 5. The features f, g in M_5 form an OR group, while they are optional in M_6 with an inclusive CC "$r \to f \vee g$". However, their PPLs are the same (see PPL$_{5,6}$ in Fig. 5).

The three lemmas above imply the following important result. Given a model M, let us call the sets of Boolean formulas $\mathcal{EX}_M$ and $\mathcal{OR}_M \cup \mathcal{IN}_M$, resp., the *exclusion* and the *mandatoriness* theories provided by M.

Theorem 9 If two feature models M and M' over the same set of features have the same partial product lines, $\mathbb{P}(M) = \mathbb{P}(M')$, then the models have the same feature tree, the exclusion theories they provide are semantically equivalent, and the mandatoriness theories they provide are semantically equivalent as well. □

Discussion. The theorem shows that the ppKS semantics for feature models accurately captures both their tree structure and constraints. The former component (that practitioners often call *feature hierarchy*) has always been a challenging issue for the Boolean semantic [37], but is well manageable with our ppKS semantics. That is why we call the ppKS semantics *faithful*. Note also that any standard complete axiomatization of the Booleans semantics allows us to replace semantic equivalence above by mutual derivability of the theories.

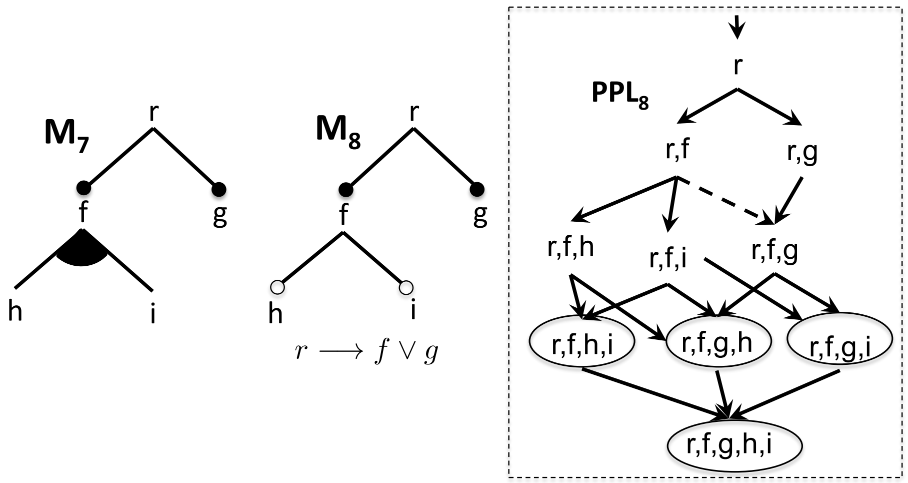


Figure 6: M_7 and M_8 are not semantically equal.

However, given two models with the same tree-structure, equivalent exclusive constraints, and the same full products, their PPLs are not necessarily identical. This fact is due to the i2c principle. As an example, consider the two models M_7 and M_8 in Fig. 6. The only difference between these models is that an equivalent inclusive CC has been used in M_8 is place of an $\mathcal{OR}$ group in M_7.

These two models satisfy the conditions (i), (ii), and (iii) in Theorem 9. However, their PPLs are not the same. PPL_8 in the figure represents $\mathbb{P}(M_8)$.

We can get the PPL of M_7 by removing the dashed transition from $\{r, f\}$ to $\{r, f, g\}$ (due to the $\mathsf{i2c}$ principle). □

It is easy to see that if we do not consider $\mathsf{i2c}$ as a mandatory requirement in generating PPLs for models, Theorem 9 above would be bidirectional:

Proposition 3 Two models over the same set of features, $M = (T, \mathcal{OR}, \mathcal{EX}, \mathcal{IN})$ and $M' = (T', \mathcal{OR}', \mathcal{EX}', \mathcal{IN}')$, are (-$\mathsf{i2c}$)-semantically equivalent, i.e., $\mathbb{P}^{-\mathsf{i2c}}(M) = \mathbb{P}^{-\mathsf{i2c}}(M')$, iff the following three conditions hold: (i) $T = T'$, (ii) $\mathcal{EX}^{\models} = \mathcal{EX}'^{\models}$, and (iii) $(\mathcal{OR} \cup \mathcal{IN})^{\models} = (\mathcal{OR}' \cup \mathcal{IN}')^{\models}$

Proof: Obviously, Theorem 9 holds also on PPLs without $\mathsf{i2c}$, as the proofs of Lemmas 3, 4, and 5 have nothing to do with $\mathsf{i2c}$. Consider two models M and M' such that (i), (ii), and (iii) hold for them. We will show that $\mathbb{P}^{-\mathsf{i2c}}(M) = \mathbb{P}^{-\mathsf{i2c}}(M)$. (i) implies that $\mathsf{BL}(T_M) = \mathsf{BL}(T_{M'})$. (ii) implies that $\mathsf{BL}(\mathcal{EX}_M)^{\models} = \mathsf{BL}(\mathcal{EX}_{M'})^{\models}$. Therefore, $(\mathsf{BL}(\mathcal{EX}_M) \cup \mathsf{BL}(T_M))^{\models} = (\mathsf{BL}(\mathcal{EX}_{M'}) \cup \mathsf{BL}(T_{M'}))^{\models}$, which implies that $\mathsf{BL}(M)^{\models} = \mathsf{BL}(M')^{\models}$. This means that, according to Proposition 2,

$\mathcal{PP}_M^{-\mathsf{i2c}} = \mathcal{PP}_{M'}^{-\mathsf{i2c}}$.

(iii) implies that $(\mathsf{BL}^!(\mathcal{OR}_M) \cup \mathsf{BL}^!(\mathcal{IN}_M))^{\models} = (\mathsf{BL}^!(\mathcal{OR}_{M'}) \cup \mathsf{BL}^!(\mathcal{IN}_M))^{\models}$. Then, considering (i) and (ii), $\mathsf{BL}^!(M)^{\models} = \mathsf{BL}^!(M')^{\models}$. Therefore,

$\mathcal{FP}_M = \mathcal{FP}_{M'}$.

Since $\mathcal{PP}_M^{-\mathsf{i2c}} = \mathcal{PP}_{M'}^{-\mathsf{i2c}}$ and $\mathcal{FP}_M = \mathcal{FP}_{M'}$, according to Proposition 2 and the maximality property of PPLs without $\mathsf{i2c}$, $\mathbb{P}^{-\mathsf{i2c}}(M) = \mathbb{P}^{-\mathsf{i2c}}(M)$. □

7 Other Applications of the Modal Logic View of Feature Modeling

In this section, we discuss some concrete tasks in feature modeling, which would benefit from the modal logic view of models.

7.1 Analysis of models

Analysis of models is an important practical issue, and as industrial models can contain thousands of features, the analysis should be automated [4]. A big group of analysis problems rely on the Boolean semantics of models. For example, given a model M, we may be interested in checking whether $PL(M)$ is not empty [39], or whether a given set of features G is a valid full product, i.e., $G \in PL(M)$ [24]. We may also be interested in finding the set of common (core) features among all full products, $\bigcap PL(M)$ [39], or checking whether f is a core feature, i.e., $f \in \bigcap PL(M)$. Specifically, an important problem is to find so called *dead* features, which do not occur in any product [24]. A typical practical approach to these analysis problems is to encode the model by a Boolean theory, and then use off-the-shelf tools like SAT-solvers [3].

However, there are some other important analysis problems, in which the use of the Boolean semantics can be error-prone. For example, it is often important to know if one model M_1 is a *refactoring* of another model M_2, or a *specialization* of M_2, or neither [38]. Standard definitions of refactoring and specialization are based on semantics, which in the Boolean case gives rise to defining refactoring $M_1 \simeq M_2$ as $PL(M_1) = PL(M_2)$ and specialization $M_1 \preceq M_2$ as $PL(M_1) \subseteq PL(M_2)$. However, as we have seen above, the Boolean semantics is too poor and makes the definitions above inadequate for their goals (see the example in the introduction). Hence, in practice, to investigate refactoring and specialization, engineers should work with pairs $(PL(M), M)$, whose second component represents the feature hierarchical structure not captured by the first component. Working with such pairs brings two issues. First, it leads to obvious maintenance problems: if one of the components changes, the user must remember to propagate the changes to the other component. Second, having a syntactical "non-Boolean" object of analysis does not allow us to use SAT (or SMT) solvers. However, the PPL semantics allows us to manage both issues. As our completeness theorem shows, $PPL(M)$ adequately captures the feature hierarchy, and hence we can analyze a single object, $PPL(M)$ or, equivalently, the modal theory $\mathsf{ML}(M)$. In Sect. 6, we have deeply discussed refactoring in the semantics sense (PPLs).

Finally, there are analysis problems only addressing the hierarchy, e.g., finding the *Lowest Common Ancestor* (LCA) of a set of features in the feature tree [29]. The PPL semantics allows us to analyze such a problem by using a model checker: given a set of features G and a candidate common ancestor feature c, we need to check whether the Kripke structure $PPL(M)$ satisfies $\bigwedge G \to c$. This way, we could get the set of common ancestors of G. Let us denote it by C. Now, to check whether an element $l \in C$ is the LCA of G, we just need to check if $PPL(M)$ satisfies $l \to \bigwedge C$. Other syntactical analysis problems can be approached in the same way: a model M is represented by a Kripke structure $PPL(M)$, the problem to be analyzed is encoded by a ppCTL-formula ϕ, and a model checker tool is used for checking if $PPL(M) \models \phi$.

7.2 PL-builder vs. PL-user View

Modal properties of product lines may not be so important for the user, for whom a model is just a structure of check-boxes to guide his choices. However, modal properties can be important for the vendor, who should plan and provide a reasonable production of all products in the product line. For example, consider the following scenario.

Suppose we want to design a chassis with two mandatory components: an engine and a frame. An engine is of type e_1 xor e_2, and a frame is of type f_1 xor f_2, as specified in the Fig. 7. In general, engine e_i better fits in frame f_i, $i = 1, 2$, but the frame supplier can modify the frame for an extra cost. Thus, we have four full products $P_0 \cup P_{ij}$ with $P_0 = \{c, e, f\}$ and $P_{ij} = \{e_i, f_j\}$, $i, j = 1, 2$ (c, e, and f stand for chassis, engine, and frame, resp.).

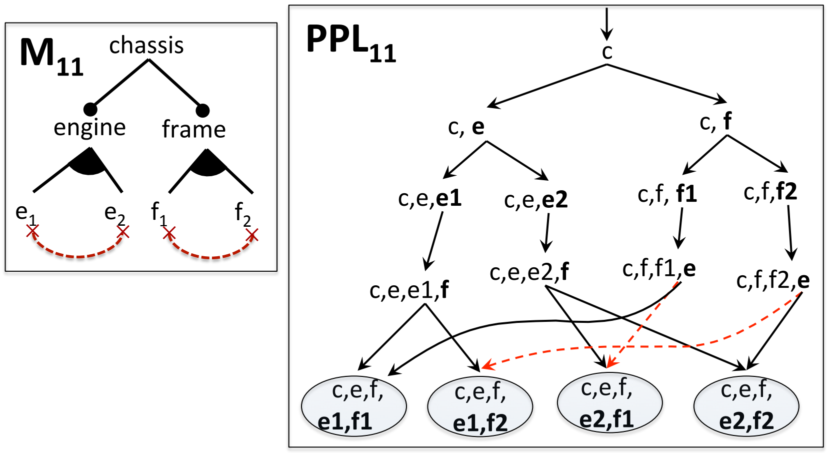


Figure 7: A model of an Engine Frame (a), and its PPL (b)

There are two ways for assembling the chassis. If we first decide on the engine type, then, for engine e_i, we may choose either to order frame f_i, or frame f_j, $j \neq i$, with a suitable modification, depending on what is cheaper (we assume that each frame type has its own supplier). Thus, from each product $P_0 \cup \{e_i\}$, $i = 1, 2$ there are two transitions as shown in Fig. 7. However, if we first decide on the frame type, then only the engine of the respective type can be mounted on the frame, and transitions from $P_0 \cup \{f_i\}$ to $P_0 \cup \{f_i, e_j\}$ $j \neq i$ are illegal (shown dashed/red in Fig. 7). To exclude the illegal transitions from the ppl, we need to add to the model the following two modal CCs: $(f_i \wedge e \wedge \neg e_i) \to \mathsf{AX}\neg e_j$ for $i, j \in \{1, 2\}$ and $i \neq j$. Such constraints cannot be expressed in BL as they do not change the *set* of partial products, and only transition are affected.

7.3 Reverse Engineering of models

Reverse engineering of models is an active research area in feature modeling. It addresses the following problem: given a PL, we want to build an appropriate model representing the PL. Depending on the PL representation, current approaches are grouped into two kinds: reverse engineering of models from (a) Boolean logic formulas [14], and (b) from textual descriptions of features [2, 30]. She et al. in [37] argue that none of these approaches is complete. Indeed, the main challenge is to recover an appropriate hierarchical structure of features. The Boolean logic approach is incomplete, since, as already discussed, the Boolean logic semantics cannot capture the feature hierarchy. The textual approach is also deficient as it is informal, and also "suggests only a single hierarchy that is unlikely the desired one" [37]. To relieve the deficiencies of these approaches, the current stat-of-the-art approach [37] proposes a heuristics-based procedure, which uses both types of input. However, if we take the given input to be the ppCTL theory of the PL, reverse engineering of models becomes simpler and more manageable, as the theory contains everything needed to build a corresponding model. Also, our careful decomposition of a model's structure and the respective theories into small blocks allows better tuning of the reverse engineering process. Our ML theories $\mathsf{ML}(T)$, $\mathsf{ML}^!(\mathcal{OR}) \cup \mathsf{ML}^!(\mathcal{IN})$, and $\mathsf{ML}(\mathcal{EX})$ capture, resp., the tree-structure T, the mandatoriness requirements ($\mathcal{OR}$-groups and inclusive crosscutting constraints), and the exclusive constraints.

8 Related Work

We discuss the connection between models and event modelling of concurrent systems in Sect. 8.1; other related work is discussed in Sect. 8.2.

8.1 Feature vs. Event-based Concurrency Modeling

In this section, we summarize similarities and differences between feature modeling and event-based concurrency modeling. We also point to several possibilities of fruitful interactions between the two disciplines.

Following the survey in [40], we distinguish three approaches in event modeling. The first is based on a topological notion of a configuration structure $(E,\mathcal{C})$ with E a (possibly infinite) set of *events*, and $\mathcal{C}\subset 2^E$ a family of subsets (usually finite) of events, which satisfy some closure conditions (e.g., under intersection and directed union). Sets from $\mathcal{C}$ are called *configurations* and understood as states of the system: $X \in \mathcal{C}$ is a state in which all events from X already occurred.

In the second approach, valid configurations are specified indirectly by some structure $\boldsymbol{D}$ of dependencies between events, which make some configurations invalid. Formally, some notion of *validity* of a set $X \subset E$ with respect to $\boldsymbol{D}$ is specified so that an *event structure* $(E,\boldsymbol{D})$ determines a configuration structure $\{X \subset E : X \text{ is } \textit{valid} \text{ with respect to } \boldsymbol{D}\}$. Typical representatives of this approach are Winskel's prime and general event structures [41], and Pratt's event spaces [33].

The third approach, originating in [20], is an ordinary encoding of sets of propositions by Boolean logical formulas. Then an event model is just a Boolean theory, i.e., a pair (E,Φ) with Φ a set of propositional formulas over set E of propositions. The left half of Table 5 summarizes this rough mini-survey.

Table 5: Event vs. feature modeling

Approach	Event Models	Feature Models	
		Boolean	Modal
Topological	$(E,\mathcal{C})$	$(F,\mathcal{PP},\mathcal{FP})$	$(F,\mathcal{PP},\to,I)$
Structural	$(E,\boldsymbol{D})$	(F,M)	
Logical	(E,Φ)	$(F,\mathsf{BL}(M),\mathsf{BL}^!(M))$	$(F,\mathsf{ML}(M))$

Importantly, transitions between states are typically considered a derived notion: in [20], any set inclusion is a transition, and in [40], special conditions are to hold in order for a set inclusion to be a valid transition. A notable exclusion is *event automata* in [31], i.e., tuples $(E,\mathcal{C},\to,I)$ with $\to$ a *given* transition relation over configurations (states), and $I\in\mathcal{C}$ an initial state.

Feature modeling is directly related to event modeling, and actually can be seen as a special interpretation of event modeling. Indeed, features can be considered as events, (partial) products as configurations, and models as special event-structures: An model $M = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ can be seen as a special encoding of a set of dependencies analogous to $\boldsymbol{D}$ (the middle row of the table). An important distinction of the Boolean feature modeling is the presence of a special subset of *final* states (products), so that feature modeling's topological and logical counterparts are triples rather than pairs (see the Boolean column in the table). Pinna and Poigné in [31] mention final states (they call them *quiescent*) but do not actually use them, whereas for feature modeling, final products are a crucial ingredient.

The last column of the table describes feature modeling's basic topological and logical structures in the modal logic view: the upper row is our notion of ppKS, and the bottom one is the theory specified in Sect. 5. Our ppKS is exactly an event automaton with quiescent states, which, additionally, satisfies the conditions of Left-totality of the transition relations and Self-loops only, but Pinna and Poigné do not apply modal logic for specifying event automata's properties (and do not even mention it); they also do not consider the i2c-principle.

The comparison above shows enough similarities and differences to hope for a fruitful interaction between the two fields. We are currently investigating what feature modeling can usefully bring to event modeling; and can mention several simple findings. The presence of two separate Boolean theories allows us to formally distinguish between *enabling* and *causality* [20]. Also, we are not aware of propositional specifications of *transient* conflicts (discussed on page 84) such as our Boolean and modal encoding of i2c. These encodings are nothing but a compact formal specification of a transaction mechanism, which is usually considered to be non-trivial.

Recently similar generalizations were proposed for event modeling in the formalism of *DCR-Graphs* [21]. DCR-Graphs employ two relations between events, *condition* (the same as the causality relation in prime event structures) and *response*, that correspond to our *subfeature* and *mandatoriness* relations, respectively. Their *markings* roughly correspond to our partial products, and *initially required response events* somehow correspond to full products. DCR-Graphs also use two additional relations *dynamic include/exclude*, which allow them to model several important constructs in concurrent distributed workflow, including transient conflicts. However, we conjecture that models provide more expressiveness for event modeling than DCR-Graphs do. The main reason is that response events (dually) correspond to maximal runs of configurations. Correspondence between response events and full products would then enforce full products to be terminal in our PPL, while there are models with some non-terminal full products (for instance, see Fig. 2). A detailed comparative analysis of models and DCR-Graphs should be an interesting research task.

These observations show that a simple feature model formalism is capable of encoding complex modal theories specifying non-trivial concurrent phenomena.

8.2 Related Work in Feature Modeling

Formal Language based Approaches. Several approaches, [3, 12, 15], and [35], have been proposed connecting feature modeling to formal languages. The closest work to ours is [35], where we provided a semantics for *cardinality-based feature models* (a generalization of models in which we deal with feature instances) by using formal languages as the semantic domain. We first proposed a generalization of *cardinality-based* FDs (CFDs), called *cardinality-based regular expression diagrams* (CRDs) in which a label of a node can be any regular expression built over a set of features. Then, a reduction process was provided going from a given CRD to a regular expression. It was proven that the regular expression generated for a given CFD captures both the full products and the hierarchy of the CFD. As for CCs, we proposed a language interpretation of them, which allowed us to integrate the semantics of CFDs and CCs over them.

The main similarity between the two approaches is that they both provide faithful semantics for feature modeling. However, they do so in two different ways. To be able to discuss their differences in detail, we transform

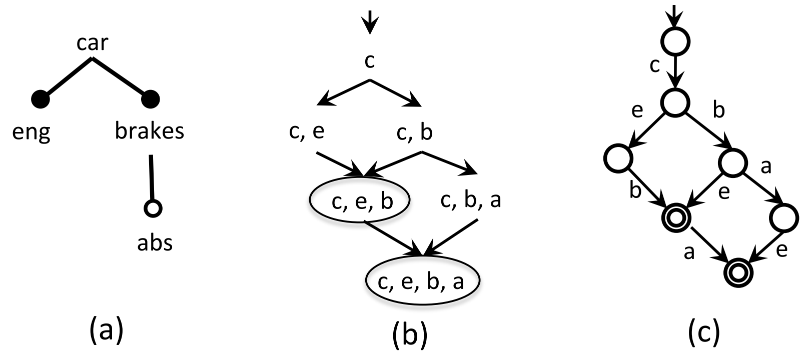


Figure 8: (a) M, (b) $\mathbb{P}(M)$, (c) $A(M)$

PPLs to automata as follows. The singletonicity property in ppKSs (see

Definition 9) allows us to transform PPLs into *finite state automata* (FSA) in a straightforward way. Indeed, there is a duality between a PPL and its corresponding FSA. Fig. 8(b) represents the PPL of the model M in Fig. 8(a)– the full products are circled. Fig. 8(c) represents the corresponding FSA of the PPL, where the final states are identified by double circles. Let $A(M)$ denote this automaton. Applying the translation procedure on M described in [35], the regular expression generated for M would be equal to $R = c\ (e\ b\ (\varepsilon + a) + b\ (\varepsilon + a)\ e)$. Note that there are infinite number of automata whose languages are equal to the language of R. On the other hand, the Kripke approach generates a unique automaton for a given model, as we saw in the example above. Roughly speaking, the Kripke approach is an *imperative* approach, while the language approach is a *declarative* one. Also, the language of $A(M)$ is not equal to the language of R. (the latter is a proper subset of the former.)

Algebraic Approaches. An algebraic model based on commutative idempotent semirings was developed in [22]. Given an model M, its PL is encoded as a term in the algebra generated by M's *leaf* features, so that non-leaf features are derived. In contrast, for us, *all* features are basic, which better conforms to a common feature modeling practice.

Amongst algebraic models for PLs, the closest to ours is a process algebra, called *PL-CCS* [26], which extends the classical CCS by an operator $\oplus$ to model variability. Each $\oplus$ occurrence in a PL-CCS expression is equipped with a unique index, and runtime occurrences with the same index must make the same choice. Processes are interpreted as products and the behaviour of a PL is given by a set of processes whose semantics is given by multi-valued Kripke structures. There are interesting similarities and differences between PL-CCS and our approach. In PL-CCS, a PL's behaviour is reconstructed from an immediate PL specification. In contrast, we extract the behaviour from the model, which we have shown can be seen as an indirect PL's specification providing everything needed to reconstruct the behavior. We might say that in PL-CCS, the expressive power of models is underestimated as they are seen in the Boolean perspective. Importantly, PL-CCS allows for recursive definitions of processes, which makes it more expressive than our ppCTL. However, allowing recursive product definitions leads us beyond the boundaries of the tree-based models and our goals in the present paper. Iterative definitions are possible in cardinality-based models. On the other hand, crosscutting constraints cannot be expressed in PL-CCS, but are readily specified in our approach (we even allow for modal CCs).

Staged Configurations. Czarnecki *et al* introduced and developed the concept of (*multi-level*) *staged configuration* in [11, 13]: given an model M, its full products are instantiated via consecutive specializations (called stages) of M by either discarding an optional feature or making it mandatory for the stage at hand and all consecutive stages. This process is continued until a fully specialized model denoting only one configuration is reached. A formal semantics for such multi-level staged configurations was defined by Classen *et al* [9]. The idea was further developed by Hubaux *et al* [23], who proposed to map models to tasks and conditions of workflows. Their approach supports parallel execution of stages and choice between them, and iterative configurations. Although both PPLs and configuration stages show how to instantiate full products, they are essentially different. Configuration paths are sequences of *models* with *decreasing variability*, whereas instantiation paths in PPLs are sequences of *products* with *increasing commonality.* Thus, the two frameworks aim at different goals and are somewhat orthogonal (but, of course, PPLs cover variability too as full products are included into PPL).

Feature Transition Systems. In a series of papers summarized in [8], Classen *et al* proposed an elegant and effective solution to checking a given pl of transition systems (TS) in a single run of a model checker rather than checking each of the TSs separately. The entire pl is encoded as a *feature TS* (FTS), in which transitions are labeled by both actions and Boolean expressions over features as Boolean variables. A truth assignment to the feature variables defines the behaviour of a single product, and the FTS as a whole represents the entire pl. They also defined a logic fCTL to allow CTL properties to refer to specific products in the line and extended the model checking procedures to support checking FTSs against fCTL properties. Their tools are capable of reporting, in a single model checking run, all products for which a property holds, as well as those for which it fails to hold. In [10], Cordy *et al* extend a common model checking framework known as CEGAR, to support FTSs as well. Thus, FTS and our idea are orthogonal ideas: for the former, a product is a TS, while for us a product is a set of features without any functional properties. These two ideas can be combined in a single formalism, but we leave it for future work.

Hierarchical Multiset Semantics. Safilian and Mibaum in [34] propose a multiset-based theory for a given CFD, which is called the *hierarchical theory* of the CFD. The theory is based on a defined hierarchy of multisets over the set of features. The hierarchical theory of the CFD is a subset of this hierarchy. It is shown that the theory captures all information about the diagram (even explicitly distinguishing between the grouped and solitary features). The theory provides a promising theoretical framework to address some challenging issues in feature model management and reverse engineering of CFDs. However, they have not managed CCs in their theory (mentioned as a future work).

9 Future Work

We describe several interesting open problems in the modal logic view of models, which would be theoretically and practically important.

(i) Complete Axiomatic System for ppCTL. Finding a sound and complete axiomatic system for ppCTL is theoretically interesting. It would be also important in practice to do automated analysis over basic feature models (see (ii) below). As we know, ppCTL is a fragment of CTL plus a constant modality !. Several sound and complete axiomatic systems have been proposed for CTL, including [17], [5], and [25]. We can take advantage of these axiomatic systems to approach a sound and complete axiomatic system for ppCTL.

(ii) Modal logic theory of Boolean semantics. Let Ψ be a Boolean theory over a set of atomic propositions F, and $PPL(\Psi) = \{P \subset F\colon\ P \models \Psi\}$ its set of models. We can consider $PPL(\Psi)$ as a *discrete* Kripke structure without transitions (and injective labeling). We can convert it into a normal Kripke structure $\mathbb{P}(\Psi)$ by considering inclusions between states, and only them, as transitions. Now for a modal formula ϕ in some modal logic ML, we write $\Psi \models^* \phi$ if $\mathbb{P}(\Psi) \models_{\mathsf{ML}} \phi$. Note that while the latter relation is an ordinary semantic entailment (for the logic ML), relation $\models^*$ is between formulas in different logics. Specifically, if our modal logic has a zero-ary modality, then we can define $(\Psi, \Psi') \models^* \phi$ for a pair of Boolean formulas such that $\Psi' \models_{\mathsf{BL}} \Psi$. These considerations show that modal logic can be employed for specifying properties of Boolean semantics, i.e., as a meta-theory for Boolean logic. We are not aware of a systematic study of this construction. For example, how could the relations $\models^*$ be axiomatized?

(iii) Automated analysis of models. To implement analysis operations over a given feature model M, one could apply either a model checker or theorem prover. To apply a model checker, we need to transform M to its PPL $\mathbb{P}(M)$ and characterize given analysis problems in terms of ppCTL formulas. We plan to implement the analysis operations over some realistic examples using existing model checking tools. To take advantage of theorem provers, we first need to have a complete axiomatic system for our logic. There exist some theorem provers such as BDDCTL [28], CTL-RP [42], and MLSolver [18], which can be used for reasoning about the CTL formulas.

(iv) Process algebras for ppKSs and models. Industrial systems are often very complex, and software companies usually design their systems by utilizing smaller systems, which themselves are produced by other companies [1]. Therefore, bigger feature models could be seen as composed from several smaller models. Hence, having a compositional way of defining complex models and their corresponding PPLs based on some algebra becomes important.

***(v) Strong version of* i2c**. Recall that the current version of the i2c principle says that two incomparable features can be included together in a partial product if at least one of them has been already completely instantiated. The current version of this principle is unavoidable, if we would like to realize a step-by-step computation; this is why ppKSs are enforced to satisfy the singletonicity condition (see Definition 9). However, in some contexts like concurrent systems, it also makes sense to consider a *stronger* version of the i2c principle: two incomparable features can be included together in a partial product if they *both* have been already completely instantiated. We plan to specify such a stronger version of the i2c-principle, in which a full product instantiation is always a transaction (which corresponds to replacing disjunction by conjunction in the definition of theory $\mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}})$, row (3) in Table 1). To address this problem, we would first need to modify the definition of ppKSs, as the singletonicity condition would not hold anymore. The logic would be the same, but the ppCTL theory of a given model satisfying the strong i2c principle changes (which makes the problem challenging).

(vi) Reverse engineering of models. In Sect. 6, we have shown that the PPL of a given model captures the tree structure, the exclusive constraints (up to equivalence), and the mandatoriness constraints (up to equivalence). Since the set of features and also the PPL are finite, finding the components of an appropriate model (a model, which is refactoring of the original model) would be algorithmic. We plan to address this problem in our future work.

10 Conclusion

We presented a novel behavioural view of models, in which a product is an instantiation process rather than its final result. We called the states of this process *partial products*, and showed that the set of partial products together with a set of (carefully defined) valid *transitions* between them can be considered as a special Kripke structure, whose properties are specifiable by a special fragment of CTL enriched with a constant modality. We called this logic ppCTL. Our main result show that a model can be considered as a compact representation of a rather complex ppCTL-theory. Thus, the logic of feature modeling is modal rather than Boolean. We have also discussed several concrete tasks in feature modeling, which would benefit from using the modal logic view of models. These tasks include refactoring of models, analysis of models, reverse engineering of models, and specification of cross-cutting constraints.

Acknowledgement

We are grateful to anonymous reviewers of our ICTAC'16 paper for useful comments and discussion. Special thanks go to the anonymous reviewers of this journal version for careful reading, numerous useful comments, suggestions, and stimulating criticism. Financial support was provided by NSERC and APC.

References

[1] M. Acher, P. Collet, P. Lahire, and R. France. Managing multiple software product lines using merging techniques. *University of Nice Sophia Antipolis. TechnicalReport, ISRN I3S/RR*, 6, 2010. URL: http://www.i3s.unice.fr/~mh/RR/2010/RR-10-06-P.LAHIRE.pdf.

[2] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler. An exploratory study of information retrieval techniques in domain analysis. In *Software Product Lines, 12th International Conference, SPLC 2008*, pages 67–76, 2008. doi:10.1109/SPLC.2008.18.

[3] D. Batory. Feature models, grammars, and propositional formulas. In *Software Product Lines, 9th International Conference, SPLC 2005*, pages 7–20. Springer Berlin Heidelberg, 2005. doi:10.1007/11554844_3.

[4] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 2010. doi:10.1016/j.is.2010.01.001.

[5] A. Bolotov and M. Fisher. A resolution method for CTL branching-time temporal logic. In *4th International Workshop on Temporal Representation and Reasoning, TIME '97*, pages 20–27. IEEE Computer Society, 1997. doi:10.1109/TIME.1997.600777.

[6] M. C. Browne, E/ M. Clarke, and O. Grümberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59(1-2):115–131, 1988. doi:10.1016/0304-3975(88)90098-9.

[7] D. Bustan and O. Grumberg. Simulation-based minimization. *ACM Transactions on Computational Logic (TOCL)*, 4(2):181–206, 2003. doi:10.1145/635499.635502.

[8] A. Classen, M. Cordy, P. Heymans, A. Legay, and P. Schobbens. Formal semantics, modular specification, and symbolic verification of product-line behaviour. *Science of Computer Programming*, 80:416–439, 2014. doi:10.1016/j.scico.2013.09.019.

[9] A. Classen, A. Hubaux, and P. Heymans. A formal semantics for multi-level staged configuration. In *Third International Workshop on Variability Modelling of Software-Intensive Systems VaMoS'09*, pages 51–60, 2009.

[10] M. Cordy, P. Heymans, A. Legay, P-Y. Schobbens, B. Dawagne, and M. Leucker. Counterexample guided abstraction refinement of product-line behavioural models. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 190–201, 2014. doi:10.1145/2635868.2635919.

[11] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. In *Software Product Lines, Third International Conference, SPLC 2004*, pages 266–283. Springer, 2004. doi:10.1007/978-3-540-28630-1_17.

[12] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005. doi:10.1002/spip.213.

[13] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005. doi:10.1002/spip.225.

[14] K. Czarnecki and A. Wasowski. Feature diagrams and logics: There and back again. In *Software Product Lines, 11th International Conference, SPLC 2007*, pages 23–34. IEEE Computer Society, 2007. doi:10.1109/SPLINE.2007.24.

[15] M. de Jonge and J. Visser. Grammars as feature diagrams. In *Workshop on Generative Programming 2002 (GP'02)*, pages 23–24. Citeseer, 2002.

[16] Z. Diskin, A. Safilian, T. Maibaum, and S. Ben-David. Modeling product lines with Kripke structures and modal logic. In *Theoretical Aspects of Computing - ICTAC 2015 - 12th International Colloquium*, pages 184–202. Springer, 2015. doi:10.1007/978-3-319-25150-9_12.

[17] E A. Emerson and J Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Computer and System Science*, 30(1):1–24, 1988. doi:10.1016/0022-0000(85)90001-7.

[18] O. Friedmann and M. Lange. A solver for modal fixpoint logics. *Electronic Notes in Theoretical Computer Science*, 262:99–111, 2010. doi:10.1016/j.entcs.2010.04.008.

[19] R. Gheyi, T. Massoni, and P. Borba. Automatically checking feature model refactorings. *Journal of Universal Computer Science*, 17(5):684–711, 2011. doi:10.3217/jucs-017-05-0684.

[20] V. Gupta and V. R. Pratt. Gages accept concurrent behavior. In *Foundations of Computer Science (FOCS'93)*, pages 62–71. IEEE, 1993. doi:10.1109/SFCS.1993.366881.

[21] T. Hildebrandt and R. Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. *arXiv preprint arXiv:1110.4161*, 2011. doi:10.4204/EPTCS.69.5.

[22] P. Höfner, R. Khédri, and B. Möller. An algebra of product families. *Software and System Modeling*, 10(2):161–182, 2011. doi:10.1007/s10270-009-0127-2.

[23] A. Hubaux, A. Classen, and P. Heymans. Formal modelling of feature configuration workflows. In *Software Product Lines, 13th International Conference, SPLC 2009*, pages 221–230, 2009.

[24] 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, DTIC Document, 1990. URL: http://www.sei.cmu.edu/reports/90tr021.pdf.

[25] M. Lange and C. Stirling. Focus games for satisfiability and completeness of temporal logic. In *16th Annual IEEE Symposium on Logic in Computer Science (LICS'01)*, pages 357–365. IEEE, 2001. doi:10.1109/LICS.2001.932511.

[26] M. Leucker and D. Thoma. A formal approach to software product families. In *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change - 5th International Symposium (ISoLA'12)*, pages 131–145. Springer, 2012. doi:10.1007/978-3-642-34026-0_11.

[27] M. Mannion. Using first-order logic for product line model validation. In *Software Product Lines, Second International Conference (SPLC'02)*, pages 176–187. Springer, 2002. doi:10.1007/3-540-45652-X_11.

[28] W. Marrero. Using BDDs to decide CTL. In *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005*, pages 222–236. Springer, 2005. doi:10.1007/978-3-540-31980-1_15.

[29] M. Mendonca, A. Wasowski, K. Czarnecki, and D. Cowan. Efficient compilation techniques for large scale feature models. In *Generative Programming and Component Engineering, 7th International Conference, GPCE 2008*, pages 13–22. ACM, 2008. doi:10.1145/1449913.1449918.

[30] N. Niu and S. Easterbrook. On-demand cluster analysis for product line functional requirements. In *Software Product Lines, 12th International Conference, SPLC 2008*, pages 87–96. IEEE, 2008. doi:10.1109/SPLC.2008.11.

[31] G. Michele Pinna and Axel Poigné. On the nature of events: Another perspective in concurrency. *Theoretical Computer Science*, 138(2):425–454, 1995. doi:10.1016/0304-3975(94)00174-H.

[32] K. Pohl, G. Böckle, and F. Van Der Linden. *Software product line engineering: foundations, principles, and techniques.* Springer, 2005.

[33] V. R. Pratt. Event spaces and their linear logic. In *Algebraic Methodology and Software Technology (AMAST'91)*, pages 3–25, 1991.

[34] A. Safilian and T. Maibaum. Hierarchical multiset theories of cardinality-based feature diagrams. In *10th International Symposium on Theoretical Aspects of Software Engineering (TASE'16)*, pages 136–143. IEEE, 2016. doi:10.1109/TASE.2016.14.

[35] A. Safilian, T. Maibaum, and Z. Diskin. The semantics of cardinality-based feature models via formal languages. In *FM'15: Formal Methods - 20th International Symposium*, pages 453–469. Springer, 2015. doi:10.1007/978-3-319-19249-9_28.

[36] P-Y. Schobbens, P. Heymans, and J-C. Trigaux. Feature diagrams: A survey and a formal semantics. In *14th IEEE International Conference on Requirements Engineering (RE'06)*, pages 139–148. IEEE, 2006. doi:10.1109/RE.2006.23.

[37] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki. Reverse engineering feature models. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*, pages 461–470. IEEE, 2011. doi:10.1145/1985793.1985856.

[38] T. Thum, D. Batory, and C. Kastner. Reasoning about edits to feature models. In *31st International Conference on Software Engineering (ICSE'09)*, pages 254–264. IEEE, 2009. doi:10.1109/ICSE.2009.5070526.

[39] P. Trinidad and A R. Cortés. Abductive reasoning and automated analysis of feature models: How are they connected?. In *Third International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'09)*, pages 145–153, 2009.

[40] Rob J. van Glabbeek and Gordon D. Plotkin. Configuration structures. In *10th Annual IEEE Symposium on Logic in Computer Science (LICS'95)*, pages 199–209, 1995. doi:10.1109/LICS.1995.523257.

[41] G. Winskel. *Event structure semantics for CCS and related languages*, pages 561–576. Springer Berlin Heidelberg, 1982. doi:10.1007/BFb0012800.

[42] L. Zhang, U. Hustadt, and C. Dixon. A refined resolution calculus for CTL. In *Automated Deduction–CADE-22*, pages 245–260. Springer, 2009. doi:10.1007/978-3-642-02959-2_20.

A Appendix. The BL and ppCTL theories of our running example (model M1 in Fig. 1)

In this section, we instantiate the general proofs in Sect. 5 with our running example data. Let us denote the model in Fig. 1 by $M = (T, \mathcal{OR}, \mathcal{EX}, \mathcal{IN})$, where T is the tree of the model, and the other three components denote the respective three structures over T. In detail, $T = (F, r, \_^{\uparrow})$, where $F = \{\mathsf{car}, \mathsf{eng}, \mathsf{gear}, \mathsf{brakes}, \mathsf{gas}, \mathsf{elec}, \mathsf{mnl}, \mathsf{atm}, \mathsf{oil}, \mathsf{abs}\}$, $r = \mathsf{car}$, and mapping $\_^{\uparrow}$ is defined as follows: $\mathsf{eng}^{\uparrow} = \mathsf{gear}^{\uparrow} = \mathsf{brakes}^{\uparrow} = \mathsf{car}$, $\mathsf{gas}^{\uparrow} = \mathsf{elec}^{\uparrow} = \mathsf{eng}$, $\mathsf{mnl}^{\uparrow} = \mathsf{atm}^{\uparrow} = \mathsf{oil}^{\uparrow} = \mathsf{gear}$, $\mathsf{abs}^{\uparrow} = \mathsf{brakes}$.

Mappings $\mathcal{OR}$ is defined as follows:
$\mathcal{OR}(\mathsf{car}) = \{\{\mathsf{eng}\}, \{\mathsf{gear}\}, \{\mathsf{brakes}\}\}$, $\mathcal{OR}(\mathsf{eng}) = \{\{\mathsf{gas}, \mathsf{elec}\}\}$, $\mathcal{OR}(\mathsf{gear}) = \{\{\mathsf{mnl}, \mathsf{atm}\}, \{\mathsf{oil}\}\}$, and $\mathcal{OR}(\mathsf{brakes}) = \varnothing$.

Finally, sets $\mathcal{EX}$ and $\mathcal{IN}$ are as follows: $\mathcal{EX} = \{\{\mathsf{elec}, \mathsf{mnl}\}, \{\mathsf{mnl}, \mathsf{atm}\}\}$, and $\mathcal{IN} = \{\mathsf{atm} \to \mathsf{abs}\}$.

A.1 The Boolean theory

According to Table 1, the Boolean theories associated with each of M's components are as follows:

The elements of $\mathsf{BL}(T)$:

$$\begin{gathered}
\top \to \mathsf{car} \\
\mathsf{eng} \to \mathsf{car}, \quad \mathsf{gear} \to \mathsf{car}, \quad \mathsf{brakes} \to \mathsf{car} \\
\mathsf{gas} \to \mathsf{eng}, \quad \mathsf{elec} \to \mathsf{eng}, \\
\mathsf{mnl} \to \mathsf{gear}, \quad \mathsf{atm} \to \mathsf{gear}, \quad \mathsf{oil} \to \mathsf{gear}, \\
\mathsf{abs} \to \mathsf{brakes}.
\end{gathered} \tag{1}$$

The elements of $\mathsf{BL}(\mathcal{EX})$:

$$\begin{gathered}
\mathsf{elec} \wedge \mathsf{mnl} \to \bot, \\
\mathsf{mnl} \wedge \mathsf{atm} \to \bot.
\end{gathered} \tag{2}$$

The elements of $\mathsf{BL}^!(\mathcal{OR})$:

$$\begin{gathered}\mathsf{car} \to \mathsf{eng}, \quad \mathsf{car} \to \mathsf{gear}, \quad \mathsf{car} \to \mathsf{brakes}, \\ \mathsf{eng} \to \mathsf{gas} \vee \mathsf{elec}, \\ \mathsf{gear} \to \mathsf{mnl} \vee \mathsf{atm}, \quad \mathsf{gear} \to \mathsf{oil}.\end{gathered} \tag{3}$$

The elements of $\mathsf{BL}^!(\mathcal{IN})$:

$$\mathsf{atm} \to \mathsf{abs}. \tag{4}$$

To obtain the theory $\mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}})$ (Boolean theory of i2c), we need to find the corresponding formulas, according to line (3) in Table 1, for the siblings, i.e., for the pairs (eng, gear), (eng, brakes), (gear, brakes), (gas, elec), (mnl, atm), (mnl, oil), and (atm, oil). Note that, for two sibling leaves f and g (i.e., $f^{\uparrow} = g^{\uparrow}$ and $f_{\downarrow} = g_{\downarrow} = \varnothing$), the corresponding formula, $f \wedge g \to f \vee g$ (since $\mathsf{BL}^{\mathsf{i2c}}(T^f_{\mathcal{OR}}) = \{\top \to f\} \equiv \{f\}$ and $\mathsf{BL}^{\mathsf{i2c}}(T^g_{\mathcal{OR}}) = \{\top \to g\} \equiv \{g\}$), is a tautology. Therefore, since gas, elec, mnl, atm, oil are leaves, the Boolean i2c formulas associated with the pairs (gas, elec), (mnl, atm), (mnl, oil), and (atm, oil) are all tautologies. Thus, the elements of $\mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}})$[15] are:

$$\begin{gathered}\mathsf{eng} \wedge \mathsf{gear} \to (\mathsf{gas} \vee \mathsf{elec}) \vee ((\mathsf{mnl} \vee \mathsf{atm}) \wedge \mathsf{oil}), \\ \mathsf{eng} \wedge \mathsf{brakes} \to (\mathsf{gas} \vee \mathsf{elec}) \vee \mathsf{brakes} \equiv \top, \\ \mathsf{gear} \wedge \mathsf{brakes} \to ((\mathsf{mnl} \vee \mathsf{atm}) \wedge \mathsf{oil}) \vee \mathsf{brakes} \equiv \top.\end{gathered}$$

Thus, $\bigwedge \mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}}) \equiv$

$$\mathsf{eng} \wedge \mathsf{gear} \to (\mathsf{gas} \vee \mathsf{elec}) \vee ((\mathsf{mnl} \vee \mathsf{atm}) \wedge \mathsf{oil}). \tag{5}$$

According to line (all$^!$) in Table 1, the BL theory of the full products of M, $\mathsf{BL}^!(M)$, is the set of all elements in (1), (2), (3), and (4). Thus, the theory $\bigwedge \mathsf{BL}^!(M)$ would be semantically equivalent to the conjunction of the following elements:

$$\begin{gathered}\mathsf{car} \wedge \mathsf{eng} \wedge \mathsf{gear} \wedge \mathsf{brakes}, \\ (\mathsf{gas} \vee \mathsf{elec}) \wedge (\mathsf{mnl} \vee \mathsf{atm}) \wedge \mathsf{oil}, \\ (\mathsf{mnl} \wedge \mathsf{atm} \to \bot) \wedge (\mathsf{elec} \wedge \mathsf{mnl} \to \bot), \\ (\mathsf{abs} \to \mathsf{brakes})\end{gathered} \tag{6}$$

According to line (all) in Table 1, the BL theory of the partial products of M, $\mathsf{BL}(M)$, is the set of all elements in (1), (2), and (5).

[15] Note that we wrote the semantically equivalent formulas, e.g., the second element would be $\mathsf{eng} \wedge \mathsf{brakes} \to ((\mathsf{gas} \vee \mathsf{elec}) \wedge \mathsf{eng}) \vee (\mathsf{brakes})$.

A.2 The ppCTL theory

According to Table 3, the semi-complete theory of $\mathbb{P}(M)$, $\mathsf{ML}_{\subseteq}(M)$, is equal to the union of $\mathsf{BL}(M)$ and the following ML theories:

The elements of $\mathsf{ML}^{!}_{\subseteq}(\mathcal{OR})$:

$$\begin{gathered}\mathsf{car} \to \Box^{!}\mathsf{eng},\quad \mathsf{car} \to \Box^{!}\mathsf{gear},\quad \mathsf{car} \to \Box^{!}\mathsf{brakes},\\ \mathsf{eng} \to \Box^{!}(\mathsf{gas} \vee \mathsf{elec}),\\ \mathsf{gear} \to \Box^{!}(\mathsf{mnl} \vee \mathsf{atm}),\quad \mathsf{gear} \to \Box^{!}\mathsf{oil}.\end{gathered} \tag{7}$$

The elements of $\mathsf{ML}^{!}_{\subseteq}(\mathcal{IN})$:

$$\mathsf{atm} \to \Box^{!}\mathsf{abs} \tag{8}$$

The elements of $\mathsf{ML}^{!}_{\subseteq}(M)$:

$$! \to \bigwedge \mathsf{BL}^{!}(M), \text{ where } \mathsf{BL}^{!}(M) \text{ can be found in (6)} \tag{9}$$

The elements of $\mathsf{ML}^{\mathsf{i2c}\nrightarrow}_{\subseteq}(T_{\mathcal{OR}})$[16]:

$$\begin{gathered}\mathsf{eng} \wedge \neg\mathsf{gear} \wedge \neg\mathsf{gas} \wedge \neg\mathsf{elec} \to \neg\mathsf{EX}\ \mathsf{gear},\\ \mathsf{gear} \wedge \neg\mathsf{eng} \wedge (\neg\mathsf{oil} \vee (\neg\mathsf{mnl} \wedge \neg\mathsf{atm})) \to \neg\mathsf{EX}\ \mathsf{eng},\\ \mathsf{eng} \wedge \neg\mathsf{brakes} \wedge \neg\mathsf{gas} \wedge \neg\mathsf{elec} \to \neg\mathsf{EX}\ \mathsf{brakes},\\ \mathsf{gear} \wedge \neg\mathsf{brakes} \wedge (\neg\mathsf{oil} \vee (\neg\mathsf{mnl} \wedge \neg\mathsf{atm})) \to \neg\mathsf{EX}\ \mathsf{brakes}\end{gathered} \tag{10}$$

According to Table 3, to get the complete theory of $\mathbb{P}(M)$ ($\mathsf{ML}(M)$) we need to add the following ML theories to the semi-complete theory ($\mathsf{ML}_{\subseteq}(M)$).

[16] According to Table 4, there is a corresponding formula for each pair of sibling features f and g. For our example, the corresponding formulas for other pairs of siblings are tautologies, e.g., take $f = \mathsf{brakes}$ and $g = \mathsf{eng}$: the corresponding formula would be $(\mathsf{brakes} \wedge \neg\mathsf{eng} \wedge \neg\mathsf{brakes} \to \neg\mathsf{EX}\ \mathsf{eng}) \equiv (\bot \to \neg\mathsf{EX}\ \mathsf{eng})$

The elements of $\mathsf{ML}^{\downarrow}_{+}(T)$:[17]

$$\begin{gathered}
\mathsf{car} \wedge \neg(\mathsf{eng} \vee \mathsf{gear} \vee \mathsf{brakes}) \to \mathsf{EX}\ \mathsf{eng} \wedge \mathsf{EX}\ \mathsf{gear} \wedge \mathsf{EX}\ \mathsf{brakes},\\
\mathsf{eng} \wedge \neg(\mathsf{gas} \vee \mathsf{elec}) \wedge \neg\mathsf{mnl} \to \mathsf{EX}\ \mathsf{elec},\\
\mathsf{eng} \wedge \neg(\mathsf{gas} \vee \mathsf{elec}) \to \mathsf{EX}\ \mathsf{gas},\\
\mathsf{gear} \wedge \neg(\mathsf{mnl} \vee \mathsf{atm} \vee \mathsf{oil}) \wedge \neg\mathsf{elec} \wedge \neg\mathsf{atm} \to \mathsf{EX}\ \mathsf{mnl},\\
\mathsf{gear} \wedge \neg(\mathsf{mnl} \vee \mathsf{atm} \vee \mathsf{oil}) \wedge \neg\mathsf{mnl} \to \mathsf{EX}\ \mathsf{atm},\\
\mathsf{gear} \wedge \neg(\mathsf{mnl} \vee \mathsf{atm} \vee \mathsf{oil}) \to \mathsf{EX}\ \mathsf{oil},\\
\mathsf{brakes} \wedge \neg\mathsf{abs} \to \mathsf{EX}\ \mathsf{abs}.
\end{gathered} \tag{11}$$

The elements of $\mathsf{ML}^{!}_{+}(M)$:

$$\bigwedge \mathsf{BL}^{!}(M) \to !, \text{ where } \mathsf{BL}^{!}(M) \text{ can be found in (6)} \tag{12}$$

The elements of $\mathsf{ML}^{\leftrightarrow}_{+}(T_{\mathcal{OR}}, \mathcal{EX})$:

$$\begin{gathered}
(\mathsf{gear} \to (\mathsf{mnl} \vee \mathsf{atm}) \wedge \mathsf{oil}) \wedge \mathsf{car} \to \mathsf{EX}\ \mathsf{eng},\\
(\mathsf{eng} \to \mathsf{gas} \vee \mathsf{elec}) \wedge \mathsf{car} \to \mathsf{EX}\ \mathsf{gear},\\
(\mathsf{eng} \to \mathsf{gas} \vee \mathsf{elec}) \wedge (\mathsf{gear} \to (\mathsf{mnl} \vee \mathsf{atm}) \wedge \mathsf{oil}) \wedge \mathsf{car} \to \mathsf{EX}\ \mathsf{brakes},\\
\mathsf{eng} \to \mathsf{EX}\ \mathsf{gas},\\
\mathsf{eng} \wedge \neg\mathsf{mnl} \to \mathsf{EX}\ \mathsf{elec},\\
\mathsf{gear} \wedge \neg\mathsf{elec} \wedge \neg\mathsf{atm} \to \mathsf{EX}\ \mathsf{mnl},\\
\mathsf{gear} \wedge \neg\mathsf{mnl} \to \mathsf{EX}\ \mathsf{atm},\\
\mathsf{gear} \to \mathsf{EX}\ \mathsf{oil},\\
\mathsf{brakes} \to \mathsf{EX}\ \mathsf{abs}.
\end{gathered} \tag{13}$$

The complete ML theory of the tree-structure, $\mathsf{ML}(T) = \mathsf{BL}(T) \cup \mathsf{ML}^{\downarrow}_{+}(T)$, would be the set of all elements in (1) and (11).

The complete ML theory of exclusive constraints, $\mathsf{ML}(\mathcal{EX}) = \mathsf{BL}(\mathcal{EX})$, would be the set of formulas in (2), which is semantically equivalent to $\mathsf{mnl} \wedge (\mathsf{elec} \vee \mathsf{atm}) \to \bot$.

The complete ML theory of the $\mathcal{OR}$ component, $\mathsf{ML}^{!}(\mathcal{OR}) = \mathsf{ML}^{!}_{\subseteq}(\mathcal{OR})$, would then be the set of formulas in (7), which is equivalent to the conjunction of the formulas $\mathsf{car} \to \Box^{!}(\mathsf{eng} \wedge \mathsf{gear} \wedge \mathsf{brakes})$, $\mathsf{eng} \to \Box^{!}(\mathsf{gas} \vee \mathsf{elec})$, and $\mathsf{gear} \to \Box^{!}(\mathsf{oil} \wedge (\mathsf{mnl} \vee \mathsf{atm}))$.

[17] The first element is the combination of three original elements of the set.

The complete ML theory of inclusive constraints, $\mathsf{ML}^{!}(\mathcal{IN}) = \mathsf{ML}^{!}_{\subseteq}(\mathcal{IN})$, is then $\{\mathsf{atm} \rightarrow \Box^{!}\mathsf{abs}\}$.

The complete ML theory of $\mathsf{i2c}$, $\mathsf{ML}^{\mathsf{i2c}}(T_{\mathcal{OR}}) = \mathsf{BL}^{\mathsf{i2c}}(T_{\mathcal{OR}}) \cup \mathsf{ML}^{\mathsf{i2c}\text{-}\nrightarrow}_{\subseteq}(T_{\mathcal{OR}})$, would be the set of all formulas in (10) and (5).

The complete ML theory of the full products, $\mathsf{ML}^{!}(M) = \mathsf{ML}^{!}_{\subseteq}(M) \cup \mathsf{ML}^{!}_{+}(M)$, would be equivalent to $! \leftrightarrow \bigwedge \mathsf{BL}^{!}(M)$, where $\mathsf{BL}^{!}(M)$ was defined in (6).

The complete ML theory of the partial products, $\mathsf{ML}^{\circ}(M) = \mathsf{BL}(M) \cup \mathsf{ML}^{\downarrow}_{+}(T) \cup \mathsf{ML}^{\leftrightarrow}_{+}(T_{\mathcal{OR}}, \mathcal{EX})$, is the set of formulas in (1), (2), (5), (11), and (13).