GSDLAB TECHNICAL REPORT

Modeling Product Lines with Kripke Structures and Modal Logic

Zinovy Diskin, Aliakbar Safilian, Tom Maibaum,
Shoham Ben-David

GSDLAB–TR 2014-08-1 August 2014

Generative Software
Development Lab

University of
Waterloo

Generative Software Development Laboratory
University of Waterloo
200 University Avenue West, Waterloo, Ontario, Canada N2L 3G1

WWW page: http://gsd.uwaterloo.ca/

Modeling Product Lines with Kripke Structures and Modal Logic

Zinovy Diskin[1,2], Aliakbar Safilian[1], Tom Maibaum[1], Shoham Ben-David[3]

[1] Department of Computing and Software,
McMaster University, Canada
zdiskin | safiliaa | maibaum@mcmaster.ca
[2] Generative Software Development Lab.,
Department of Electrical and Computer Engineering,
University of Waterloo, Canada
zdiskin@gsd.uwaterloo.ca
[3] University of Waterloo, Canada
shohambd@gmail.com

Abstract. Product lines (PLs) are now an established framework for software design. They are specified by special diagrams called *feature models (FMs)*. For formal analysis, FMs are usually encoded by propositional theories with Boolean semantics. We discuss a major deficiency of this semantics, and show that it can be fixed by considering a product an instantiation process rather than its final result. We call intermediate states of this process *partial products*, and argue that what an feature model M really defines is a *partial product line*, *PPL(M)*. We argue that such PPLs can be viewed as special Kripke structures (we say *feature KSs*) specifiable by a suitable version of CTL (*feature CTL*). We show that any feature model M can be represented by an fCTL theory $Th(M)$, and prove that for any fKS K, $K \models Th(M)$ iff $K = PPL(M)$.

Key words: Product Lines, Feature Models, Partial Product Lines, Kripke Structures, Modal Logic, Propositional Logic

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 optionally having some *optional* features that allow the user (or developer) to configure the product the user wants (e.g., an MS Office, a Photoshop, or a Linux kernel). Instead of producing a multitude of separate products, the vendor designs a single PL encompassing the variety of products, which results in a significant reduction in development time and cost [21]. (An impatient reader might look at the next section for a simple PL example.)

Industrial PLs may be based on thousands of features inter-related in complex ways [19]. Methods of specifying PLs and checking the validity of a PL against a specification is an active research area. The most common method

for modeling PLs is *feature modeling*. Elite software engineering conferences like ICSE, ASE, and FM, readily accept papers on PL [1,24,28]; there are conference series specially devoted to PL, such as Generative Programming and Component Engineering (GPCE) and Software Product Line Conference (SPLC), which attracts dozens of papers; and several textbooks have been written about PL and FM, e.g., [7, 21].

To manage their design and analysis, feature models (FMs) should be represented as formal objects processable by tools. A common approach is to consider features as atomic propositions, and view FMs as theories in Boolean propositional logic (**BL**), whose valid valuations are to be exactly the valid products defined by the FM [3]. This approach gave rise to a series of prominent applications for analysis of industrial size PLs [13, 27]. However, in the paper we discuss a major deficiency of this semantics, which limits the effectiveness of the approach, and show that it can be fixed by considering a product as an instantiation *process* rather than its final result. We call intermediate states of this process *partial products*, and argue that what an feature model M really defines is a *partial product line*, *PPL(M)*. We then show that any PPL can be viewed as an instance of a special type of Kripke structure (KS), which we axiomatically define and call *feature KS (fKS)*. The latter are specifiable by a suitable version of modal logic, which we call *feature CTL (fCTL)*, as it is basically a fragment of CTL enriched with a zero-ary modality that only holds in states representing final products. We show that any feature model M can be represented by an fCTL theory $Th(M)$ accurately specifying M's intended semantics: the main result of the paper states that for any fKS K, $K \models Th(M)$ iff $K = PPL(M)$. Then we can replace FMs by the respective **fCTL**-theories, which are normally well amenable to formal analysis and automated processing.

In a broader perspective, the paper aims to show that mathematical foundations of FM are mathematically interesting, and to attract the attention of the ML community to the area. We will describe several problems that we believe are mathematically interesting and practically useful. On the other hand, we would like to have the paper readable by a PL researcher, and to convince her that the logic of PL is modal rather than Boolean. Therefore, we pay special attention to the motivation of our framework: we want first to validate the mathematical model, and then explore it formally.

Our plan for the paper is as follows. The next section aims to motivate the formal framework developed in the paper. In Sect. 2.1, we discuss the basics of FM, and why its Boolean semantics is deficient. Then in Sect. 2.2 we introduce partial products and PPLs. We will begin with PPLs generated by simple FMs, which can be readily explained in lattice-theoretic terms (Sect. 2.2). Then we show that PPLs generated by complex FMs are more naturally, and even necessarily, best characterized as transition systems (Sect. 2.2). In Sect. 3, the notions of FM and the PPL it generates are formalized. In Sect. 4, we introduce the notion of fKS as an immediate abstraction of PPLs, and fCTL as a language to specify fKS properties. We show, step-by-step, how to encode FMs into **fCTL**

theories, and prove our main result. Related work is discussed in Sect. 5, and Sect. 6 concludes.

2 Feature Models and Partial Product Lines

2.1 Basics of Feature Modeling

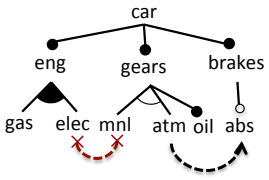
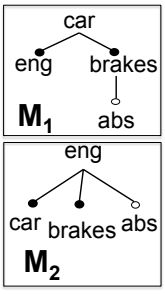


Fig. 1: An FM

A common method of specifying a PL is to build a *feature model* (FM): a graphical structure presenting a hierarchial decomposition of features with *cross-cutting constraints* (CCCs) 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-end edge says that brakes can *optionally* be equipped with abs (ABS). Black angles denote so called *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 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 big set of 2^9 possible combination of 9 features (the root is always included), and shows dependencies between choices.

Industrial FMs may have thousands of features, and their PLs can be quite complex [19]. To manage their design and analysis, FMs should be represented as formal objects processable by tools. A common approach is to consider features as atomic propositions, and view FMs as theories in Boolean propositional logic (**BL**), whose valid valuations are to be valid products defined by the FM [3]. This approach gave rise to a series of prominent applications for analysis of industrial size PLs [13, 27].



However, the **BL** encoding of FMs has an inherent drawback. The inset figure shows two essentially different FMs, which nevertheless have the same valid products: $P_1 = \{\mathsf{car}, \mathsf{eng}, \mathsf{brakes}\}$ and $P_2 = P_1 \cup \{\mathsf{abs}\}$ (and so will have the same **BL**-encoding). Hence, either (a) FMs contain redundant information, irrelevant for PLs, or (b) their Boolean semantics is too poor to capture all relevant information contained in FMs. The possibility (a) does not hold as model M_2 is evidently pathological wrt. the natural meaning of features, and we do want to distinguish M_1 and M_2. Hence, it is the case (b) that creates the discrepancy, and so the Boolean semantics, and **BL**-encoding of FMs, should be corrected. In particular, loosing the hierarchical structure of FMs in their **BL**-encoding can invalidate some important automated analyses performed by **BL**-based tools,

e.g., determining the least common ancestor of a given set of features (LCA) [4]. In general, the deficiency of **BL** is known [27], but, surprisingly, no logic has been proposed to replace **BL** to fix the problem.

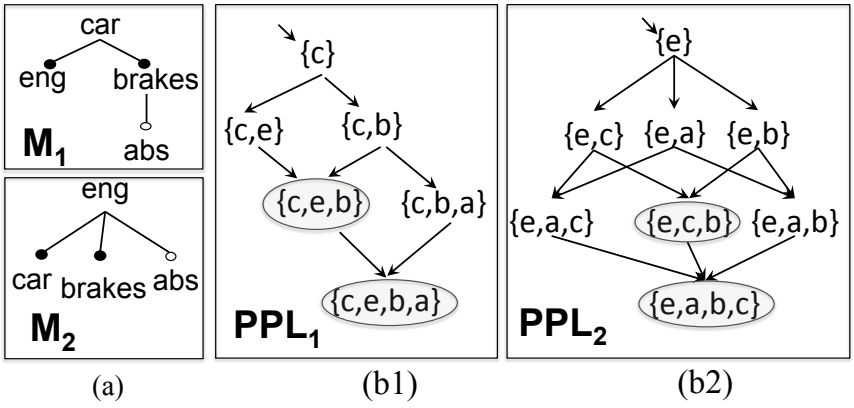


Fig. 2: From FM to PPL: simple cases

2.2 PPL semantics for FM

Getting started What is lost in the **BL**-encoding is the *dynamic* nature of the notion of product. An FM 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 will 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 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 hooked arrows and write, e.g., $\{\mathsf{c}\} \hookrightarrow_{\mathsf{e}} \{\mathsf{c}, \mathsf{e}\}$ 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 $PPL(M_1)$ or PPL_1. In a similar way the PPL of the second feature model, $PPL(M_2)$, is built in Fig. 2(b2). We see that although

both FMs have the same set of *full* products (i.e., are Boolean semantics equivalent), their PPLs are essentially different and properly capture the difference between the FMs.

Generations of $PPL_{1,2}$ 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 up-join semi-lattices). Valid products are up-closed sets of features (filters), and form distributive lattices (consider Fig. 2(b1,b2) as Hasse diagrams), whose up-join is set intersection, and down-join (meet) is set union. If we freely add meets to posets $M_{1,2}$ (eng $\wedge$ brakes etc.), and thus freely generate lattices L_i, $i = 1, 2$, over the respective posets, then lattices L_i and PPL_i will be dually isomorphic (Birkhoff duality).

The forgotten mandatoriness of some features appears as incompleteness of some objects, which form a subset of proper partial products. Its complement is the set of all full products. Thus, PPLs of simple FMs as in Fig. 2(a) are their filter lattices with distinguished subsets of full products. In the next section we will discuss whether this lattice-theoretic view works for more complex FMs.

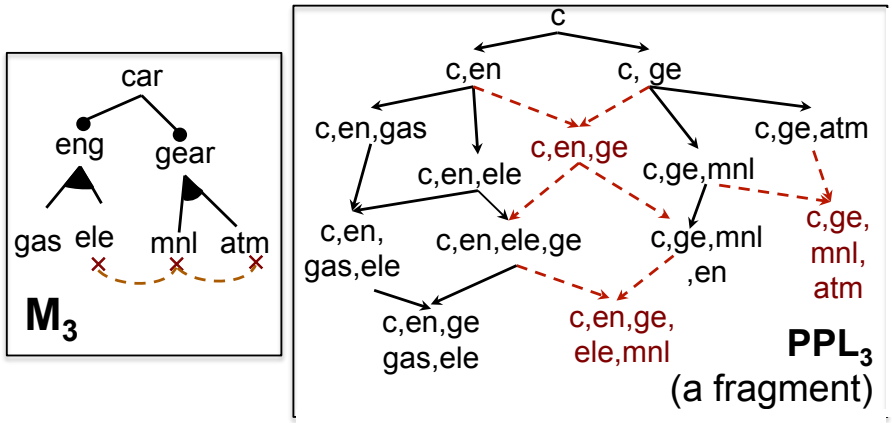


Fig. 3: From FMs to PPLs: Complex case

PPL: From lattices to transition systems Figure 3 (left) shows a fragment of the FM in Fig. 1, in which, for uniformity, we have presented the XOR-group as an OR-group with a new cross-cutting constraint (CCC) added to the tree (note the $\times$-ended arc between mnl and atm). 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 (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 FM influences the PPL. Two CCCs 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 the universal *bottom* element $\perp$ (a constant to be a common 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-union of products {c,en,ele,ge} and {c,ge,mnl,en} "blows up" and becomes equal to the set of all features including $\perp$ ("False implies everything".) The same happens with the other pair of conflicting products. (A detail discussion can be found in [22].)

Next we consider the mandatoriness structure of model M_3 (given by black bullets and triangles). Of course, this structure determines a full product subsect in the PPL (not shown in Fig. 3) as before. However, now 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, $\{\mathsf{c}\} \hookrightarrow_{\mathsf{en}} \{\mathsf{c}, \mathsf{en}\} \hookrightarrow_{\mathsf{ge}} P$ 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 is poor design practice that should be prohibited, and the corresponding transition is declared invalid. Similarly, transition $\{\mathsf{c}, \mathsf{ge}\} \hookrightarrow_{\mathsf{en}} P$ is also not valid 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). 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.

"Diagonal" transition $\{\mathsf{c}, \mathsf{ge}\} \longrightarrow \{\mathsf{c}, \mathsf{en}, \mathsf{ge}\}$ 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.

It follows from this observation that a PPL can be richer than its lattice of partial products; transition exclusion cannot be explained lattice-theoretically with Boolean logic; transition systems/Kripke structures and modal logic are needed. Moreover, even if all inclusions are transitions, the 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. All in all, the transition relation is an important (and independent) component of the general PPL structure.

Finally, as soon as transitions become first-class citizens, it makes sense to distinguish full products by supplying them (and only them) with identity loops (see the bottom product in PPL_4 in Fig. 4). Such a loop does not add (nor remove) any feature from the product, and has a clear semantic meaning: the instantiation process can stay in a full product state indefinitely.

In the next two sections, we will formalize the constructs discussed above, and prove several results about them.

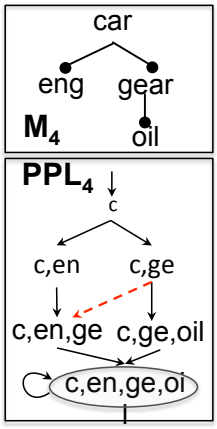


Fig. 4

3 FMs and PPLs formally

3.1 Feature Trees and Models

Typical FMs are trees with an extra structure like in Fig. 1. Non-root features are either solitary or grouped. Solitary features are either *mandatory* (e.g., eng, gear, brakes in Fig. 1) or *optional* (like abs). Feature groups are usually either OR-groups (e.g., {gas,elec}) or XOR-groups ({mnl,atm}). An FM is a feature tree with a set of additional *cross-cutting* constraints (CCCs) on features in different branches of the tree. Typically, such constraints are either *exclusive* (the x-ended arc in Fig. 1), or *inclusive* (the dashed arrow arc 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 an additional exclusive constraint as we did for model M_3 in Fig. 3(a).

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

(i) $T = (F, r, \_^{\uparrow})$ *is a tree whose nodes are called features:* F *denotes the set of all features,* $r \in F$ *is the root, and function* $\_^{\uparrow}$ *maps each non-root feature* $f \in F_{\_r} \stackrel{\text{def}}{=} F \setminus r$ *to its parent* $f^{\uparrow}$*. The inverse function that assigns to each feature the set of its children (we say* subfeatures*) is denoted by* $f_{\downarrow}$*; this set is empty for leaves. The set of all ancestors and all descendants of a feature* f *are denoted by* $f^{\uparrow\uparrow}$ *and* $f_{\downarrow\downarrow}$*, resp.*

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* $\{G \subset F\colon\ f \# g \text{ for all } f, g \in G\} \subset 2^F$*.*

(ii) $\mathcal{OR}$ *is a function that assigns to each feature* $f \in F$ *a set* $\mathcal{OR}(f) \subset 2^{f_{\downarrow}}$ *(perhaps, 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*. We write* $M(f) \subseteq f_{\downarrow}$ *for the set of all such subfeatures. For example, in Fig. 1,* $\mathcal{OR}(\mathsf{gear}) = \{\{\mathsf{mnl}, \mathsf{atm}\}, \{\mathsf{oil}\}\}$*, and* $\mathcal{OR}(\mathsf{brakes}) = \varnothing$*.*

Element in set $O(f) \stackrel{\text{def}}{=} f_{\downarrow} \setminus \bigcup \mathcal{OR}(f)$ *are called* optional *subfeatures.* □

Definition 2. (Feature Models). *A feature model* (FM) *is a triple* $M = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ *with an FT* $T_{\mathcal{OR}}$ *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) $\mathcal{IN} \subset \#2^F F$ *is a set of* inclusive dependencies *between features. For FM in Fig. 1,* $\mathcal{IN} = \{(\{\mathsf{atm}\}, \mathsf{abs})\}$.

Dependencies are also called cross-cutting constraints (CCCs).

Thus, an FM 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 quadruple* $M = (T, \mathcal{OR}, \mathcal{EX}, \mathcal{IN})$. *If needed, we will subscript* M*'s components with index* $_M$, *e.g., write* F_M *for the set of features* F.

The class of all FMs over the same feature set F *is denoted by* $\mathcal{M}(F)$. □

3.2 Full and Partial Products

A common approach for formalizing the full products of a given FM M is to use propositional logic [3]. In this approach, the features are considered as atomic propositions and a logical formula is generated by modeling relationships between features. The relationships between features are into the following kinds: subfeature relationship, OR groups, exclusive constraints, and inclusive constraints. Accordingly, we give the propositional theories $\Phi(T)$, $\Phi^!(\mathcal{OR})$, $\Phi(\mathcal{EX})$, $\Phi(\mathcal{IN})$, respectively. The theory of the full products, denoted by $\Phi^!(M)$, is the union of the above theories. What this logical translation is meant to achieve is the following equivalence: P is a valid full product wrt. an FM M iff $P \models_{\mathbf{BL}} Th(M)$, where subscript **BL** refers to the Boolean logic validity.

In this section, we show that the propositional theory of FMs can be more richer: we give a propositional theory for identifying the partial products of a given FM. To this end, we give a propositional theory of **I2C** principle and also use the theories $\Phi(T)$, $\Phi(\mathcal{EX})$.

Remark 1. As seen in Sect. 2.2, transition exclusion cannot be explained with Boolean logic and modal logic is needed. Indeed, in Sect. 4.3, we will build several ML theories of FMs analogous to their propositional theories and show how the corresponding theories change in their ML versions.

Definition 3. (From FM to Propositional Theories). *Let* $M = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ *be an FM. The upper part of Table 1 lists four propositional theories (1) - (4) determined by* M *(*$\top$ *and* $\bot$ *denote True and False, respectively.* $\bigvee G$ *and* $\bigwedge G$ *are conjunction and disjunction of all formulas in a set of formulas* G*).*[1] *We call these theories* basic.

Theories (12), (13$^!$*) and (123*$^!$*4) are unions of the respective basic theories; they are introduced to ease writing and reading technicalities.* □

[1] Theory (3$^!$) is superscripted by ! because later we will introduce another theory determined by the $\mathcal{OR}$-structure.

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

(1)	$\Phi(T) = \{\top \to r\} \cup \{f' \to f\colon\ f \in F, f' \in f_{\downarrow}\}$
(2)	$\Phi(\mathcal{EX}) = \{\bigwedge G \to \bot\colon\ G \in \mathcal{EX}\}$
(3$^!$)	$\Phi^!(\mathcal{OR}) = \{f \to \bigvee G\colon\ f \in F, G \in \mathcal{OR}(f)\}$
(4)	$\Phi(\mathcal{IN}) = \{\bigwedge G \to f\colon\ (G, f) \in \mathcal{IN}\}$
(12)	$\Phi(T_{\mathcal{EX}}) = \Phi(T) \cup \Phi(\mathcal{EX})$
(13$^!$)	$\Phi^!(T_{\mathcal{OR}}) = \Phi(T) \cup \Phi^!(\mathcal{OR})$
(123$^!$4)	$\Phi^!(M) = \Phi^!(T_{\mathcal{OR}}) \cup \Phi(\mathcal{EX}) \cup \Phi(\mathcal{IN})$
(3*)	$\Phi^{\mathbf{I2C}}(\mathcal{OR}) = \left\{f \wedge g \to (\bigwedge \Phi^!(T^f_{\mathcal{OR}})) \vee (\bigwedge \Phi^!(T^g_{\mathcal{OR}}))\colon\ f, g \in F, f^{\uparrow} = g^{\uparrow}\right\}$
(123*)	$\Phi(M) = \Phi(T_{\mathcal{EX}}) \ \cup\ \Phi^{\mathbf{I2C}}(\mathcal{OR})$

Definition 4. (Full Products). *A* full product *over feature model $M = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ is a set of features $P \subseteq F$ satisfying the theory $\bigwedge \Phi^!(M)$ defined in Table 1 as theory (123$^!$4).*

The set of all full products is called the full product set *over M and denoted by $\mathcal{FP}_M$. Thus, $\mathcal{FP}_M = \left\{P \subseteq F\colon\ P \models_{\mathbf{BL}} \bigwedge \Phi^!(M)\right\}$.* □

Remark 2. The definition above is equivalent to the standard one, except that we use the term *full* product rather than product (see below).

Our next goal is to formalize the notion of partial product as described in Sect. 2. We need to define compliance of a product with the **I2C**-principle.

Definition 5. (From FM to Propositional Theories Cont'd). *Let $T_{\mathcal{OR}} = (T, \mathcal{OR})$ be an FT. We first define the notion of a* feature subtree induced *by feature $f \in F$: it is a pair $T^f_{\mathcal{OR}} = (T^f, \mathcal{OR}^f)$ with T^f being the tree under f, i.e., $T^f \stackrel{\text{def}}{=} (f_{\downarrow\downarrow} \cup \{f\}, f, \_^{\uparrow})$, and mapping $\mathcal{OR}^f$ is inherited from $\mathcal{OR}$, i.e., for any $g \in f_{\downarrow\downarrow}$, $\mathcal{OR}^f(g) = \mathcal{OR}(g)$.* □

Next we define propsitional theory $\Phi^{\mathbf{I2C}}(\mathcal{OR})$ as specified in Table 1 in row (3*). Theory $\Phi^{\mathbf{I2C}}(\mathcal{OR})$ formalizes the following idea. If a valid product contains two incomparable features, then at least one of these features must be fully instantiated within the product.

Definition 6. (Partial Products). *A* partial product *over feature model $M = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ is a set of features $P \subseteq F$ satisfying the theory $\bigwedge \Phi(M)$ specified in Table 1 row (123*). We denote the set of all partial products by $\mathcal{PP}_M$. Thus, $\mathcal{PP}_M = \{P \subseteq F\colon\ P \models_{\mathbf{BL}} \bigwedge \Phi(M)\}$.*

Below the term 'product' will mean 'partial product'. □

Note that theory $\Phi(M)$ defining the set of partial products does not contain inclusive and $\mathcal{OR}$ constraints in contrast to theory $\Phi^!(M)$. Thus, the $\mathcal{OR}$ and $\mathcal{IN}$ components of an FM only affect the set of full products, but do not influence partial product formation.

Proposition 1. *For any FM M, $\Phi^!(M) \models \Phi(M)$. Hence, full products as defined in Definition 4 form a subset of partial products, $\mathcal{FP}(M) \subseteq \mathcal{PP}(M)$.* □

Proof. Obviously, $\Phi^!(\mathcal{OR}) \models \{f \rightarrow \bigwedge \Phi^!(T^f_{\mathcal{OR}}) : f \in F\}$.
$\Rightarrow \Phi^!(\mathcal{OR}) \models \Phi^{\mathbf{I2C}}(\mathcal{OR})$.
$\Rightarrow \Phi(T) \cup \Phi^!(\mathcal{OR}) \cup \Phi(\mathcal{EX}) \cup \Phi(\mathcal{IN}) \models \Phi(T) \cup \Phi(\mathcal{EX}) \cup \Phi^{\mathbf{I2C}}(\mathcal{OR})$.
$\Rightarrow \Phi^!(M) \models \Phi(M)$.

3.3 PPLs as Transition Systems

In this section, we consider how 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 $P' = P \uplus \{f\}$ is valid as well. We then write $P \longrightarrow P'$ and call the pair (P, P') *a valid (elementary) 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, $P' \models \Phi(T_{\mathcal{EX}})$. Compatibility with **I2C** is more complicated.

Definition 7. (Relative fullness). *Given a product P and a feature $f \notin P$, the following theory (continuing the list in Table 1) is defined:*

$$(6)_{P,f} \qquad \Phi^{\mathbf{I2C}}(P, f) \stackrel{\text{def}}{=} \bigcup \{\Phi^!(T^g_{\mathcal{OR}}) \colon\ g \in P \cap (f^\uparrow)_\downarrow\}$$

where $T^g_{\mathcal{OR}}$ denotes the subtree induced by feature g as described in Definition 5. (Note that set $P \cap (f^\uparrow)_\downarrow$ may be empty, and then theory $\Phi^{\mathbf{I2C}}(P, f)$ is also empty.)

We say that P is fully instantiated wrt. feature f *if $P \models \Phi^{\mathbf{I2C}}(P, f)$* □

For example, it is easy to check that for FM M_4 in Fig. 4, for product $P_1 = \{\mathsf{car}, \mathsf{eng}\}$ and feature $f_1 = \mathsf{gear}$, we have $P_1 \models \Phi^{\mathbf{I2C}}(P_1, f_1)$ while for $P_2 = \{\mathsf{car}, \mathsf{gear}\}$ and $f_2 = \mathsf{eng}$, $P_2 \not\models \Phi^{\mathbf{I2C}}(P_2, f_2)$ because $\Phi^!(T^{\mathsf{gear}}_{\mathcal{OR}}) = \{\mathsf{gear} \rightarrow \mathsf{oil}\}$ and $P_2 \not\models \{\mathsf{gear} \rightarrow \mathsf{oil}\}$.

Definition 8. (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 two conditions hold: (a1) $P' \models \Phi(T_{\mathcal{EX}})$, and (a2) $P \models \Phi^{\mathbf{I2C}}(P, f)$.

(b) $P' = P$ and then P is full. □

Proposition 2. *If P is a valid product and $P \longrightarrow P'$, then P' is also a valid product.* □

Proof. Let P be a valid product and $P \longrightarrow P'$ be a valid transition.
According to Definition 8, there are two choices: (a) $P = P'$ or (b) $P' = P \uplus \{f\}$ s.t. $P' \models \Phi(T_{\mathcal{EX}})$ and $P \models \Phi^{\mathbf{I2C}}(P, f)$.
Since the proposition is proven immediately in the former case, we consider the later case.
Let $g \in P$ be an arbitrary feature with $g^\uparrow = f^\uparrow$, i.e., $g \in P \cap (f^\uparrow)_\downarrow$. Since

$P \models \Phi^{\mathbf{I2C}}(P, f)$, $P \models \Phi^{!}(T^{g}_{\mathcal{OR}})$. This implies that $P' \models \Phi^{!}(T^{g}_{\mathcal{OR}})$.
Therefore, $P' \models \bigcup \{\Phi^{!}(T^{g}_{\mathcal{OR}}) : (g \in P) \wedge (g^{\uparrow} = f^{\uparrow})\}$.
The above statement, along with $P \models \Phi^{\mathbf{I2C}}(\mathcal{OR})$, imply that

$$P' \models \left\{ f \wedge g \rightarrow (\bigwedge \Phi^{!}(T^{f}_{\mathcal{OR}})) \vee (\bigwedge \Phi^{!}(T^{g}_{\mathcal{OR}})) \colon \ f, g \in F, f^{\uparrow} = g^{\uparrow} \right\},$$

i.e., $P' \models \Phi^{\mathbf{I2C}}(\mathcal{OR})$.
Thus P' satisfies both $\Phi(T_{\mathcal{EX}})$ and $\Phi^{\mathbf{I2C}}(\mathcal{OR})$, which means $P' \models \Phi(M)$. The proposition is proven.

Definition 9. (Partial Product Line). *Let $M = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ be an FM. The* partial product line *determined by M, $\mathbb{P}(M)$, is a tuple $(\mathcal{PP}_M, \longrightarrow_M, I_M)$, with the set $\mathcal{PP}_M$ of partial products equipped with all valid transitions $\longrightarrow_M$ as defined above (so that full products, and only they, are equipped with self-loops), and the initial product $I_M = \{r_M\}$ consisting of the root feature.* □

4 Feature Kripke Structures and Modal Logic

In this section, we introduce *Feature Kripke Structures* (fKS): an immediate abstraction of PPLs generated by FMs in terms of Kripke structures. Then (in section 4.2), we introduce a modal logic called *feature CTL* (**fCTL**), which is tailored for specifying fKSs' properties. Section 4.3 presents the main results: we translate an FM into an **fCTL** theory specifying the intended PPL.

4.1 Feature Kripke Structures

We deal with a special type of Kripke structures, in which possible worlds (called partial products) are identified with sets of atomic propositions (features), and hence the labeling function is not needed.

Definition 10. (Feature Kripke Structure (fKS)). *Let F be a finite set (of* features*). An* fKS *over F is a tuple $K = (\mathcal{PP}, \longrightarrow, I)$ with $\mathcal{PP} \subset 2^F$ a set of* (partial) products*, $I \in \mathcal{PP}$ the* initial *singleton product (i.e., $I = \{f\}$ for some $f \in F$), and $\longrightarrow \subseteq \mathcal{PP} \times \mathcal{PP}$ a binary totally defined (we say* left-total*)* transition *relation. In addition, we require the following three conditions to hold.*
*(*Singletonity*)* $\forall P, P' \in \mathcal{PP}$ *⟨if $P \longrightarrow P'$ and $P \neq P'$, then $P' = P \cup \{f\}$ for some $f \notin P$⟩.*
*(*Reachability*) For all $P \in \mathcal{PP}$, P is reachable from I.*
*(*Self-Loops Only*) For all $P, P' \in \mathcal{PP}$, if $(P \longrightarrow^{+} P' \longrightarrow^{+} P)$, then $P = P'$ (where $\longrightarrow^{+}$ is the transitive closure of $\longrightarrow$). Thus, every loop is a self-loop.*

A product $P \in \mathcal{PP}$ with $P \longrightarrow P$ is called full*. The set of full products is denoted by $\mathcal{FP}$. Ingredients of an fKS K are subscripted with $_K$ if needed ($\mathcal{PP}_K$, $\longrightarrow_k$ etc.). We denote the class of all fKSs built over set F by $\mathcal{K}(F)$.* □

Note that any product in an fKS eventually evolves into a full product because set F is finite, $\longrightarrow$ is left-total, and all loops are self-loops. The following proposition follows obviously.

Proposition 3. *Let $M \in \mathcal{M}(F)$ be an FM. Its PPL is an fKS, i.e., $\mathbb{P}(M) \in \mathcal{K}(F)$.* □

The proposition above is not very interesting: there is a rich structure in $\mathbb{P}(M)$ that is not captured by the fact that $\mathbb{P}(M)$ is an fKS—the class $\mathcal{K}(F)$ is too big. We want to characterize $\mathbb{P}(M)$ in a more precise way by defining as small as possible class of fKSs to which $\mathbb{P}(M)$ would provably belong. Hence, we need a logic for defining classes of fKSs by specifying fKS's properties.

4.2 Feature Computation Tree Logic

We define **fCTL**, which is a fragment of the Computation Tree Logic (CTL) enriched with a constant (zero-ary) modality ! to denote full products.

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

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

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

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 $\forall \langle P{=}P_1 \longrightarrow P_2 \longrightarrow \ldots \rangle \ \exists i \geq 1.\ P_i \models \phi$
$P \models \mathsf{AG}\phi$	iff $\forall \langle P{=}P_1 \longrightarrow P_2 \longrightarrow \ldots \rangle \ \forall i \geq 1.\ P_i \models \phi$
$P \models !$	iff $P \longrightarrow P$

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

Then we set $K \models \phi$ iff $I_K \models \phi$. Also, we say a class $\mathcal{K}$ of fKSs satisfies a given formula ϕ, and write $\mathcal{K} \models \phi$, iff $K \models \phi$ for all $K \in \mathcal{K}$.

4.3 fCTL-theory of Feature Models

Given an arbitrary model M, we aim to build an **fCTL**-theory *Th(M)* implicitly encoded by M. The goal is achieved if for an arbitrary fKS K we have $K \models Th(M)$ iff $K = \mathbb{P}(M)$. Then we can replace FMs by the respective **fCTL**-theories, which are normally well amenable to formal analysis and automated processing.

Table 3: Definitions of (basic) fCTL theories extracted from model $M = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$

(1)	$\Phi^{\downarrow}_{\mathrm{ML}}(T) = \{f \wedge \neg \bigvee f_{\downarrow} \rightarrow \mathsf{EX} g : f, g \in F, g^{\uparrow} = f\}$
(2)	$\varnothing$
($3^{!}$)	$\Phi^{!}_{\mathrm{ML}}(\mathcal{OR}) = \{f \rightarrow \Box^{!} \bigvee G \colon\ f \in F, G \in \mathcal{OR}(f)\}$
(3^{*})	$\Phi^{\mathbf{I2C}\rightarrow}_{\mathrm{ML}}(\mathcal{OR}) = \left\{f \wedge \neg \bigwedge \Phi^{!}(T^{f}_{\mathcal{OR}}) \rightarrow \neg \mathsf{EX} g \colon\ f, g \in F, f^{\uparrow} = g^{\uparrow}\right\}$
(23)	$\Phi^{\leftrightarrow}_{\mathrm{ML}}(T_{\mathcal{OR}}, \mathcal{EX}) = \{\bigwedge \Phi^{\mathbf{I2C}}(f) \wedge \neg f \wedge \neg \bigvee \Phi^{\mathcal{EX}}(f) \rightarrow \mathsf{EX} f : f \in F\}$, where $\Phi^{\mathbf{I2C}}(f) = \{g \rightarrow \Phi^{!}(T^{g}_{\mathcal{OR}}) : g, f \in F, g^{\uparrow} = f^{\uparrow}, g \neq f\}$ and $\Phi^{\mathcal{EX}}(f) = \{\bigwedge(G \setminus \{f\}) : G \in \mathcal{EX}, f \in G\}$
(4)	$\Phi_{\mathrm{ML}}(\mathcal{IN}) = \{\bigwedge G \rightarrow \Box^{!} f \colon\ (G, f) \in \mathcal{IN}\}$

We will build a required theory by composing multiple small *elemetary* or *basic* **fCTL**-theories, each of which specifies a fragment of M's structure. Roughly, this process is analogous to how we build Boolean theories encoded by FMs (Table 1), but, of course, the modal setting makes it much more complex and richer. Table 3 defines several basic **fCTL**-theories; their indexes are chosen to establish some loose parallelism with Table 1. Table 4 specifies the necessary composed theories, which gradually provide the necessary fKS-structure.

We can summarize the results of this section in the following way:

"*An fKS K satisfies the theory $Th(M)$ iff it is equal to the PPL of M.*"

To prove this statement, we need to show that the following statements hold:

(i) "The PPL of M satisfies the theory $Th(M)$" (Lemma 5).

(ii) "The set of p-products of K and M are the same" (Lemma 6).

(iii) "The set of transitions of K and $\mathbb{P}(M)$ are the same" (Lemma 8).

Several interesting propositions and lemmas are proven in turn to support the above lemmas. Some of them address directly some practical analysis operation regarding to relations between FMs including *specialization* and *refactoring*.

Proposition 4. *Let K and M be an fKS and an FM over the same set of features, respectively. $K \models \mathsf{AG} \bigwedge \Phi(M)$ iff $\mathcal{PP}_K \subseteq \mathcal{PP}_M$.* □

Proof. Proof of $\big(K \models \mathsf{AG} \bigwedge \Phi(M)\big) \Rightarrow \big(\mathcal{PP}_K \subseteq \mathcal{PP}_M\big)$:
$K \models \mathsf{AG}\, \Phi(M)$ implies that for any product P in K, P satisfies $\Phi(M)$. According

Table 4: Definitions of composed fCTL theories extracted from model $M = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$

BL Theories (see Table 1)	**fCTL** Theories (see Table 3)
$\Phi(T)$	$\Phi_{\text{ML}}(T) = \Phi(T) \cup \Phi^{\downarrow}_{\text{ML}}(T)$
$\Phi(T_{\mathcal{EX}})$	$\Phi_{\text{ML}}(T_{\mathcal{EX}}) = \Phi_{\text{ML}}(T) \cup \Phi(\mathcal{EX})$
$\Phi(M)$	$\Phi_{\text{ML}}(M) = \Phi_{\text{ML}}(T_{\mathcal{EX}}) \cup \Phi^{\leftrightarrow}_{\text{ML}}(T_{\mathcal{OR}}, \mathcal{EX}) \cup \Phi^{\mathbf{I2C}}(\mathcal{OR})$
$\Phi^{!}(T_{\mathcal{OR}})$	$\Phi^{!}_{\text{ML}}(T_{\mathcal{OR}}) = \Phi_{\text{ML}}(T) \cup \Phi^{!}_{\text{ML}}(\mathcal{OR})$
$\Phi^{!}(M)$	$\Phi^{!}_{\text{ML}}(M) = \Phi^{!}_{\text{ML}}(T_{\mathcal{OR}}) \cup \Phi_{\text{ML}}(\mathcal{IN}) \cup \{\bigwedge \Phi^{!}(M) \rightarrow !\}$
$\Phi^{\mathbf{I2C}}(\mathcal{OR})$	$\Phi^{\mathbf{I2C}}_{\text{ML}}(\mathcal{OR}) = \Phi^{\mathbf{I2C}}(\mathcal{OR}) \cup \Phi^{\mathbf{I2C}\rightarrow}_{\text{ML}}(\mathcal{OR})$
N/A	$Th(M) = \Phi_{\text{ML}}(M) \cup \Phi^{\mathbf{I2C}}_{\text{ML}}(\mathcal{OR}) \cup \Phi^{!}_{\text{ML}}(M)$

to Definition 6, this means that any product in K is a partial product of M, i.e., $\forall P \in \mathcal{PP}_K : \ P \in \mathcal{PP}_M$.

Proof of $\big(\mathcal{PP}_K \subseteq \mathcal{PP}_M\big) \Rightarrow \big(K \models \mathsf{AG} \bigwedge \Phi(M)\big)$:
Let $\mathcal{PP}_K \subseteq \mathcal{PP}_M$ and $P \in \mathcal{PP}_K$. Since $P \in \mathcal{PP}_M$, according to Definition 6, P satisfies $\bigwedge \Phi(M)$. Since any state in K is reachable from the initial state, $I_K \models \mathsf{AG} \ \bigwedge \Phi(M)$, which means $K \models \mathsf{AG} \ \bigwedge \Phi(M)$.

The following is an immediate corollary of the above proposition:

Corollary 1. *Given an FM M, $\mathbb{P}(M) \models \mathsf{AG} \ \bigwedge \Phi(M)$.*

Lemma 1. *Given an FM M, $\mathbb{P}(M) \models \mathsf{AG} \ \bigwedge \Phi^{\downarrow}_{\text{ML}}(T)$.* □

Proof. Recall that $\Phi^{\downarrow}_{\text{ML}}(T) = \big\{f \wedge \neg \bigvee f_{\downarrow} \rightarrow \mathsf{EX}g : f, g \in F, g^{\uparrow} = f\big\}$.

Let f be a feature and $P \in \mathcal{PP}_M$ a partial product of M such that $f \in P$ and for any feature $g \in f_{\downarrow}$ (g is a subfeature of f), $g \notin P$. In other words, $P \models f \wedge \neg \bigvee f_{\downarrow}$. We want to show that for any subfeature of f, say g, $P \models \mathsf{EX}g$, i.e., there is a partial product P' such that $P \longrightarrow_M P'$ and $g \in P'$:

Since P is a partial product of M, according to Definition 6, $P \models \Phi(\mathcal{EX}) \wedge \Phi(T)$. Let $P' = P \cup \{g\}$.

Since exclusive constraints are defined on incomparable features (Definition 2 - recall that two features h, i are comparable if $h \in i_{\downarrow\downarrow}$ or $i \in h_{\downarrow\downarrow}$), adding g to P does not violate $\Phi(\mathcal{EX})$. Therefore, $P' \models \Phi(\mathcal{EX})$. It is also clear that adding g to P does not violate $\Phi(T)$: $P \models \Phi(T)$ and $g^{\uparrow} \in P$ and so $P' \models \Phi(T)$. Satisfying both $\Phi(\mathcal{EX})$ and $\Phi(T)$, $P' \models \Phi(T_{\mathcal{EX}})$.

Since $P \cap (g^{\uparrow})_{\downarrow} = \varnothing$ (note that $g^{\uparrow} = f$ and all children/subfeatures of f are absent in P), $\Phi^{\mathbf{I2C}}(P', g) = \varnothing$ (note Definition 7). Therefore, $P \models \Phi^{\mathbf{I2C}}(P, f)$.

We showed above that $P' \models \Phi(T_{\mathcal{EX}})$ and $P \models \Phi^{\mathbf{I2C}}(P, f)$. According to Definition 8, there is a transition $P \longrightarrow_M P'$. Hence, according to Proposition 2, P' is a partial product of M, i.e., $P' \in \mathcal{PP}_M$. The lemma is proven.

Lemma 2. *Given an FM M, $\mathbb{P}(M) \models \mathsf{AG}\ \bigwedge \Phi^{\leftrightarrow}_{\mathrm{ML}}(T_{\mathcal{OR}}, \mathcal{EX})$.* □

Proof. Recall that $\Phi^{\leftrightarrow}_{\mathrm{ML}}(T_{\mathcal{OR}}, \mathcal{EX}) = \big\{ \bigwedge \Phi^{\mathbf{I2C}}(f) \wedge \neg f \wedge \neg \bigvee \Phi^{\mathcal{EX}}(f) \rightarrow \mathsf{EX} f : f \in F \big\}$, where $\Phi^{\mathbf{I2C}}(f) = \{ g \rightarrow \Phi^{!}(T^{g}_{\mathcal{OR}}) : g, f \in F, g^{\uparrow} = f^{\uparrow}, g \neq f \}$ and $\Phi^{\mathcal{EX}}(f) = \big\{ \bigwedge (G \setminus \{f\}) : G \in \mathcal{EX}, f \in G \big\}$.

Let f and P be a feature and a partial product of M, respectively, such that $P \not\models f$, $P \models \bigwedge \Phi^{\mathbf{I2C}}(f)$, and $P \not\models \bigvee \Phi^{\mathcal{EX}}(f)$, i.e.,

(a) $f \notin P$,

(b) for any feature g with $g^{\uparrow} = f^{\uparrow}$, if $g \in P$, then $P \models \Phi^{!}(T^{g}_{\mathcal{OR}})$, and

(c) $P \cup \{f\}$ does not violate any exclusive constraints.

(b) implies that $P \cup \{f\}$ does not violate the **I2C** principle, i.e., $P \models \Phi^{\mathbf{I2C}}(P, f)$, and (c) implies that $P \cup \{f\}$ does not violate the exclusive constraints, i.e., $P \cup \{f\} \models \Phi(T_{\mathcal{EX}})$. Thus, according to Definition 8, there exists a transition $P \longrightarrow_M P \cup \{f\}$, which implies $P \models \mathsf{EX}\ f$.

The above result in $\mathbb{P}(M) \models \mathsf{AG}\ \bigwedge \Phi^{\leftrightarrow}_{\mathrm{ML}}(T_{\mathcal{OR}}, \mathcal{EX})$.

Lemma 3. *Given an FM M, $\mathbb{P}(M) \models \mathsf{AG}\ \bigwedge \Phi^{!}_{\mathrm{ML}}(\mathcal{OR})$.* □

Proof. Recall that $\Phi^{!}_{\mathrm{ML}}(\mathcal{OR}) = \big\{ f \rightarrow \Box^{!} \bigvee G \colon\ f \in F, G \in \mathcal{OR}(f) \big\}$.

Let f, G, and P be a feature, a subset of features, and a partial product, respectively, such that $G \in \mathcal{OR}(f)$ and $f \in P$.

Since $\mathbb{P}(M)$ is an fKS (Proposition 3), $\longrightarrow_M$ is left-total and the Singletonity and Self-Loops Only conditions hold in $\mathbb{P}(M)$. According to left-totality of $\longrightarrow_M$ and the Self-Loops Only condition, there exists a partial product P' with $P \longrightarrow^{*}_{M} P'$ and $P' \longrightarrow_M P'$ (P' is a full product accessible from P).

According to Singletonity condition, $f \in P'$. Since any full product including f must include also some of the features in G, the statement $G \cap P' \neq \varnothing$ holds. This implies $\mathbb{P}(M) \models \mathsf{AG}\ \bigwedge \Phi^{!}_{\mathrm{ML}}(\mathcal{OR})$

Lemma 4. *Given an FM M, $\mathbb{P}(M) \models \mathsf{AG}\ \bigwedge \Phi_{\mathrm{ML}}(\mathcal{IN})$.* □

Proof. Recall that $\Phi_{\mathrm{ML}}(\mathcal{IN}) = \big\{ \bigwedge G \rightarrow \Box^{!} f \colon\ (G, f) \in \mathcal{IN} \big\}$.

Let f, G, and P be a feature, a subset of features, and a partial product, respectively, such that $(G, f) \in \mathcal{IN}$ and $P \models \bigwedge G$.

Consider a partial product P' with $P \longrightarrow^{*}_{M} P'$ and $P' \longrightarrow_M P'$ (P' is a full product accessible from P).

According to Singletonity condition, $G \subset P'$. Since any full product superset of G must include f, $P' \models f$. This implies that $\mathbb{P}(M) \models \mathsf{AG}\ \bigwedge \Phi_{\mathrm{ML}}(\mathcal{IN})$.

Proposition 5. *Given an FM M, $\mathbb{P}(M) \models \mathsf{AG}\ \bigwedge \Phi^{\mathbf{I2C}\rightarrow}_{\mathrm{ML}}(\mathcal{OR})$.* □

Proof. Recall that $\Phi^{\mathbf{I2C}\rightarrow}_{\mathrm{ML}}(\mathcal{OR}) = \big\{ f \wedge \neg \bigwedge \Phi^{!}(T^{f}_{\mathcal{OR}}) \rightarrow \neg \mathsf{EX} g : f, g \in F, f^{\uparrow} = g^{\uparrow} \big\}$.

Assume by way of contradiction that $\mathbb{P}(M) \not\models \mathsf{AG}\ \bigwedge \Phi^{\mathbf{I2C}\rightarrow}_{\mathrm{ML}}(\mathcal{OR})$. Then there are two features f, g, and two partial products P, P' such that $f \in P$, $P \not\models \bigwedge \Phi^{!}(T^{f}_{\mathcal{OR}})$ (the feature f in not completely disassembled in P), $P \longrightarrow_M P'$, $g^{\uparrow} = f^{\uparrow}$, and $g \in P'$.

Thus, P' violates the **I2C** principle, i.e., $P' \not\models \Phi^{\mathbf{I2C}}(\mathcal{OR})$. This leads us to a contradiction.

Now, we can prove one of our main lemmas, called *the satisfaction lemma*, stating that the partial product of a given FM M satisfies its corresponding **fCTL**-theory:

Lemma 5 (The Satisfaction Lemma). *For any FM M, $\mathbb{P}(M) \models \mathsf{AG} \bigwedge Th(M)$.*

Proof. (a) $\mathbb{P}(M) \models \mathsf{AG} \bigwedge \Phi_{\mathrm{ML}}(T)$ is an immediate corollary of Lemma 1 and Corollary 1.

(b) $\mathbb{P}(M) \models \mathsf{AG} \bigwedge \Phi_{\mathrm{ML}}(T_{\mathcal{EX}})$ is an immediate corollary of (a) and Corollary 1.

(c) $\mathbb{P}(M) \models \mathsf{AG} \bigwedge \Phi_{\mathrm{ML}}(M)$ is an immediate corollary of (b), Lemma 2, and Corollary 1.

(d) $\mathbb{P}(M) \models \mathsf{AG} \bigwedge \Phi^{!}_{\mathrm{ML}}(T_{\mathcal{OR}})$ is an immediate corollary of (a) and Lemma 3.

(e) It is clear that any product satisfying $\bigwedge \Phi^{!}(M)$ is a full product, i.e., $\mathbb{P}(M) \models \mathsf{AG}\ (\bigwedge \Phi^{!}(M) \rightarrow !)$. This, along with Lemma 3 and Lemma 4, result in $\mathbb{P}(M) \models \mathsf{AG} \bigwedge \Phi^{!}_{\mathrm{ML}}(M)$.

(f) $\mathbb{P}(M) \models \mathsf{AG} \bigwedge \Phi^{\mathbf{I2C}}_{\mathrm{ML}}(\mathcal{OR})$ is an immediate corollary of Proposition 5 and Corollary 1.

$\mathbb{P}(M) \models \mathsf{AG} \bigwedge Th(M)$ is an immediate corollary of (c), (e), and (f).

Working in the opposite direction is much more difficult; it is given by a sequence of lemmas proven in the following, and culminates with the main theorem. In the following, F is a set of features, $M \in \mathcal{M}(F)$ and $K \in \mathcal{K}(F)$.

Lemma 6 (The Products Lemma). *$K \models \mathsf{AG} \bigwedge \Phi_{\mathrm{ML}}(M)$ implies $\mathcal{PP}_K = \mathcal{PP}_M$.*

Proof. Since $K \models \mathsf{AG}\ \Phi(M)$, according to Proposition 4, $\mathcal{PP}_K \subseteq \mathcal{PP}_M$. Now, we need to show that $\mathcal{PP}_M \subseteq \mathcal{PP}_K$:

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

We do a pre-order depth-first traversal of T_P of a special kind complying **I2C**-priniciple: in each level of the tree, all the nodes that are completely disassembled must be visited before the other nodes. 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 output of the traversal would be the sequence $\langle \mathsf{car}, \mathsf{gear}, \mathsf{mnl}, \mathsf{oil}, \mathsf{eng} \rangle$. Let $S_P = \langle f_1, \ldots, f_n \rangle$, where $f_1 = r$, be the result of the traversal of T_P.

The following condition (R) holds:

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

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

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

(*base case*): $K \models \mathsf{AG}\ r$ implies that $I_K \models r$ (I_K denotes the initial state of K). Since I_K is a singleton set, $I_K = \{r\} = \{f_1\}$.

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

(*inductive step*): We want to prove that any prefix of the sequence $\langle f_1, \ldots f_i, f_{i+1}\rangle$ is a partial product of 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 prove this for both cases (R-1) and (R-2), as introduced above:

(R-1):

Since f_i is freshly added to the state P' and f_{i+1} is a subfeature of f_i ($f_{i+1}^{\uparrow} = f_i$), due to $K \models \mathsf{AG}\ \bigwedge \Phi^{\downarrow}_{\mathrm{ML}}(T)$, there is a transition $P' \longrightarrow_K P' \cup \{f_{i+1}\}$. Therefore, $\{f_1, \ldots, f_{i+1}\} \in \mathcal{PP}_K$.

(R-2):

Since $\forall g \in P' : (g^{\uparrow} = f_{i+1}^{\uparrow}) \Rightarrow (P' \models \Phi^{!}(T^{g}_{\mathcal{OR}}))$ (note (R-2) above), $P' \models \Phi^{\mathbf{I2C}}(f_{i+1})$.

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

Since $P' \models \Phi^{\mathbf{I2C}}(f_{i+1}) \wedge \neg \bigvee \Phi^{\mathcal{EX}}(f_{i+1}) \wedge \neg f_{i+1}$, and $K \models \Phi^{\leftrightarrow}_{\mathrm{ML}}(T_{\mathcal{OR}}, \mathcal{EX})$, there is a partial product $\{f_1, \ldots, f_{i+1}\} \in \mathcal{PP}_K$ such that $P' \longrightarrow_K P' \cup \{f_{i+1}\}$.

Therefore, $P \in \mathcal{PP}_K$, which means that $\mathcal{PP}_M \subseteq \mathcal{PP}_K$. The lemma is proven.

Lemma 7. *Given an FM M and an fKS K, $K \models \mathsf{AG}\ \bigwedge(\Phi^{!}_{\mathrm{ML}}(M) \cup \Phi_{\mathrm{ML}}(M))$ implies*

(i) $\forall P \in \mathcal{PP}_M.\ (P \longrightarrow_M P) \Rightarrow (P \longrightarrow_K P)$.
(ii) $\forall P \in \mathcal{PP}_K.\ (P \longrightarrow_K P) \Rightarrow (P \longrightarrow_M P)$. □

Proof. Recall that $\Phi^{!}_{\mathrm{ML}}(M) = \Phi^{!}_{\mathrm{ML}}(T_{\mathcal{OR}}) \cup \Phi_{\mathrm{ML}}(\mathcal{IN}) \cup \{\bigwedge \Phi^{!}(M) \rightarrow !\}$. Since $K \models \mathsf{AG} \bigwedge \Phi_{\mathrm{ML}}(M)$, according to Lemma 6, $\mathcal{PP}_K = \mathcal{PP}_M$.

(i):

$K \models \mathsf{AG}\ \big(\bigwedge \Phi^{!}(M) \rightarrow !\big)$ and $\mathcal{PP}_K = \mathcal{PP}_M$ together imply that for any partial product $P \in \mathcal{PP}_M$ with $P \longrightarrow_M P$ (P is a full product of M), there is a self-loop on P in K, i.e., $P \longrightarrow_K P$ (P is a full product in K).

(ii):

Let $P \in \mathcal{PP}_K$ such that $P \longrightarrow_K P$. To show that $P \longrightarrow_M P$ (P is a full product of M), we need to prove $P \models \Phi^{!}(M)$. Recall that $\Phi^{!}(M) = \Phi^{!}(\mathcal{OR}) \cup \Phi(T) \cup$

$\Phi(\mathcal{EX}) \cup \Phi(\mathcal{IN})$. From $K \models \Phi_{\text{ML}}(M)$ we can infer that $K \models \Phi(T) \cup \Phi(\mathcal{EX})$ (note Table 4). In the following, we show that $P \models \Phi^{!}(\mathcal{OR}) \cup \Phi(\mathcal{IN})$:

Recall that $\Phi^{!}(\mathcal{OR}) = \{f \rightarrow \bigvee G\colon\ f \in F, G \in \mathcal{OR}(f)\}$. Let f be a feature, and $G \in \mathcal{OR}(f)$, and $f \in P$. $P \models \bigwedge \Phi^{!}_{\text{ML}}(\mathcal{OR})$ implies $P \models (f \rightarrow \Box^{!} \bigvee G)$ (note Table 3). Since $f \in P$ and $P \longrightarrow_K P$, $P \models \bigvee G$. Therefore, $P \models \Phi^{!}(\mathcal{OR})$.
Recall that $\Phi(\mathcal{IN}) = \{\bigwedge G \rightarrow f\colon\ (G, f) \in \mathcal{IN}\}$. Let f be a feature and $G \subset P$ such that $(G, f) \in \mathcal{IN}$. Note that $P \models \Phi_{\text{ML}}(\mathcal{IN})$ implies $P \models \bigwedge G \longrightarrow \Box^{!} f$. Since $G \subset P$ and $P \longrightarrow P$, $f \in P$. Therefore, $P \models \Phi(\mathcal{IN})$.

Based on the above, $P \longrightarrow_M P$.

Lemma 8 (The Transition Lemma). $K \models \mathsf{AG}\ \bigwedge Th(M)$ *implies* $\longrightarrow_K = \longrightarrow_M$.

Proof. According to Lemma 6, $K \models \mathsf{AG}\ \bigwedge \Phi_{\text{ML}}(M)$ implies $\mathcal{PP}_M = \mathcal{PP}_K$. 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 8, $P' \models \Phi(T_{\mathcal{EX}})$, and $P \models \bigwedge \Phi^{\mathbf{I2C}}(P, f)$. Thus, there are two choices:

(i) $\Phi^{\mathbf{I2C}}(P, f) = \varnothing$

(ii) $\Phi^{\mathbf{I2C}}(P, f) \neq \varnothing$

(i): This implies that the parent of f is freshly added through a transition ingoing to P. Hence, due to $K \models \mathsf{AG}\ \Phi^{\downarrow}_{\text{ML}}(T)$, there exists a transition $P \longrightarrow_K P'$.
(ii): Since $P' \models \Phi(\mathcal{EX})$, $P \models \neg \bigvee \Phi^{\mathcal{EX}}(f)$. Also, $P \models \bigwedge \Phi^{\mathbf{I2C}}(P, f)$ implies that $P \models \Phi^{!}(T^{g}_{\mathcal{OR}})$ for any $g \in P \cap (f^{\uparrow})_{\downarrow}$, which means $P \models \bigwedge \Phi^{\mathbf{I2C}}(f)$. Hence, due to $\Phi^{\leftrightarrow}_{\text{ML}}(T_{\mathcal{OR}}, \mathcal{EX})$, 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. Due to Lemma 7, any loop transition in $\mathbb{P}(M)$ is also a loop transition in K. Thus, $\longrightarrow_M \subseteq \longrightarrow_K$. Indeed, there may be some illegal transitions in K due to **I2C** principle.
The theory $\Phi^{\mathbf{I2C}\rightarrow}_{\text{ML}}(\mathcal{OR})$ excludes all illegal transitions due to **I2C**. Hence, $\longrightarrow_M = \longrightarrow_K$.

The following theorem, *the main theorem*, shows how to characterize the partial product line of a given FM using an **fCTL**-theory. Indeed, it is an immediate corollary of the lemmas 5, 6 and 8.

Theorem 4.15 [The Main Theorem] $K \models \mathsf{AG}\ \bigwedge Th(M)$ iff $K = \mathbb{P}(M)$.

5 Related Work

5.1 Feature Modeling Languages

There are many feature modelling languages. Some of them, including the classical FODA [17], are tree-based, e.g., [8,12], others are DAG-based such as [18,23]. In this paper, we have restricted our attention to feature trees, and do not consider cardinality-based feature modeling [10], in which the same feature can occur in the product multiple times. To see how we deal with cardinality-based FMs,

please refer to [25]. In the current section, we survey different FM languages and compare them with our own definition of FMs discussed in Sect. 3.1.

Feature Oriented Domain Analysis (FODA: 1990) [17] is a tree-based language where there exists only an XOR type for grouped edges. In this language, cross-cutting constraints (CCCs) are expressed textually. Fig. 5 gives an example. Note in the figure that mandatory edges are represented by ordinary edges. FORM [18] is the DAG version of FODA.

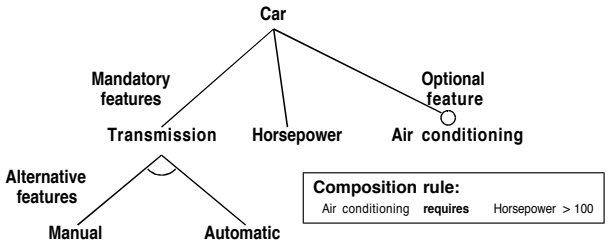


Fig. 5: FODA: adopted from [17]

Feature Reuse-Driven Software Engineering Business (FeatuRSEB: 1998) [14] is a DAG-based language and, in addition to XOR, it supports OR grouped edges. CCCs in this language are presented graphically. Note that graphical representations for CCCs make them less expressive, since CCCs can involve only two features. Fig. 6 provides an example.

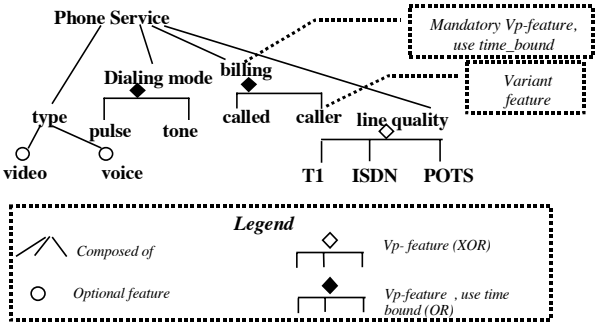


Fig. 6: FeatuRSEB: adopted from [14]

Generative Programming (GP: 2000) [8] adapts FMs in the context of generative programming. Their FDs are simply FODA with the addition of OR grouped edges.

Van Gurp et al. (2001) [32] extend FeatuRSEB to deal with binding times, indicating when features can be selected, and external features, which are technical possibilities of the system. Binding times are used to annotate relationships between features and external features are represented in dashed boxes. These

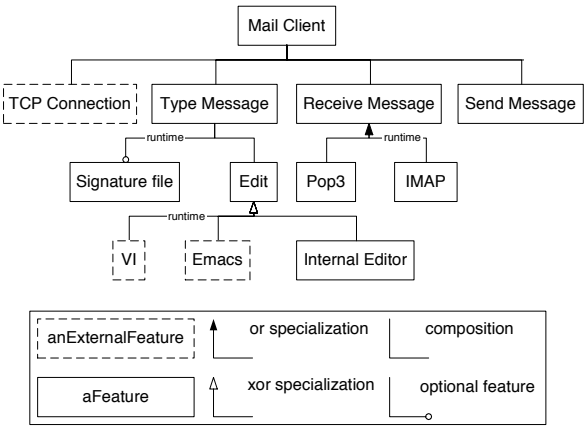


Fig. 7: van Grup et al: adopted from [32]

changes mainly concern concrete syntax, and certainly have no influence on feature combinations. Fig. 7 gives an example.

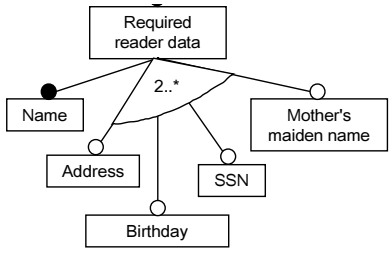


Fig. 8: PLUSS: adopted from [23]

Riebisch et al. (2002) in [23] propose a DAG-based language where grouped edges are replaced by UML-like multiplicities. In their work, CCCs are presented graphically. Fig. 8 provides an example.

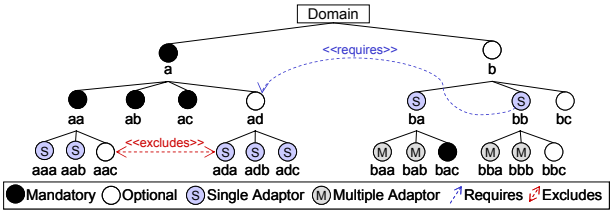
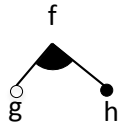


Fig. 9: PLUSS: from [12]

The Product Line Use case modelling for System and Software engineering (PLUSS: 2005) method [12] combines FMs and use case diagrams to depict a

high level view of a product line. It is a tree-based language where the usual FM representation conventions are changed: single nodes with a circled 'S' represent an XOR-decomposition of their parent while multiple nodes with a circled 'M' represent OR-decomposition. They use graphical notations for CCCs. Fig. 9 gives an example.



As mentioned above, many languages have been introduced for modeling PLs. However, they never received a formal semantics until Schobbens et al. in 2007 released their work "generic semantics of feature diagrams" [26]. This lack of formality makes some problems such as ambiguity: "combinations of mandatory and optional features with OR and XOR relations could lead to ambiguities" [23]. As an example, consider the inset figure. This FM requires the choice in two steps. Based on the OR-relation a non-empty subset of $\{g, h\}$ needs to be chosen. Then, we can choose again due to optionality on g, contrary to the original intention of the OR-relation. Note that in our syntactical definition of feature trees, Definition 1, an FD is a pair $(T, \mathcal{OR})$ in which optional nodes are derivable: a node is optional if and only if it is in the set $F - \{f \in F : f \notin \bigcup \mathcal{OR}\}$. In this way, optional nodes and grouped nodes are strictly distinct, which makes feature diagrams semantically unambiguous.

Schobbens et al. introduced a generic definition of the syntax and semantics of feature diagrams. The new language is called *free feature models* (FFMs). Indeed, they separate concrete syntax (what the user sees), abstract syntax (ignores the visual stuffs in feature diagrams that are useless to assign a formal semantics), and semantics.

Definition 12 (Free Feature Diagrams - adopted from [26]). *An FFM is a tuple $(N, P, r, \lambda, DE, \Phi)$ where N is a set of nodes, $P \subseteq N$ is its set of primitive nodes, $r \in N$ is the root, λ is a total function assigning a decomposition operation on a given node, $DE \subseteq N \times N$ representing the edges, Φ represents textually the cross-cutting constraints (this can be seen a boolean formula).* [2]

Primitive nodes are those that convey the features that are included in at least one the products and the non-primitive ones are the nodes that are included in non of the products. Let us call non-primitive nodes *dummy features*. A node cannot be assigned to more than one decomposition operations, since λ is defined as a function. This is why dummy features are included in the above definition. Fig. 10 illustrates this: to convert the FM M to an FFM M', we need a dummy feature f. Although this definition formalizes all FM languages in only one abstract syntax, using dummy features makes it critical in the sense of computational complexity and succinctness.

In [26], the set $\{\{A\}, \{B\}, \{A, B\}\}$, where A and B are features, is used as a counterexample to show that classical tree-based languages are not fully expressive. If we return to the original intuition of what a product line is, we

[2] in the original definition there is another element denoted by CE representing graphical CCCs. Since textual constraints subsume graphical ones we deliberately ignored this element.

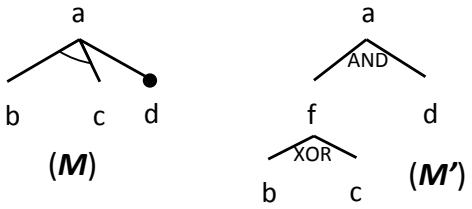


Fig. 10: From FMs to FFMs

find that this statement is not correct: "products in a product line must share some features". Given this assumption about product lines, all classical languages have the same expressiveness.

5.2 Feature Modeling vs. Concurrency Modeling

We will begin with a short primer on event-based models for concurrency. Then we discuss striking similarities between feature models and event based models, and what the differences are.

The event-based view of concurrency is well-established. A vast family of mathematical structures was developed, encompassing, amongst others, event structures (ES) by Winskel [33], gates (or event Kripke structures, EKS) by Gupta and Pratt [15], event automata (EA) by Pinna and Poigné [20], and configuration structures (CS) by van Glabbeek and Plotkin [30]. The inset table below-left summarizes abbreviations. Note that in event-based models each event is labeled with an *action* represents an occurrence of that action during a possible run of the system.

ES	Event Structure
EA	Event Automata
EKS	Event Kripke Structure
CS	Configuration Structure
EAQ	EA with Quiescent states

At the core of the framework is a topological notion of a configuration system $\mathbb{C} = (E, \mathcal{C})$ with E a set of *events* (perhaps, infinite), and $\mathcal{C} \subseteq 2^E$ a set of subsets of events called *configurations* [3]. A configuration $X \in \mathcal{C}$ is understood as a state of the system, in which all events from X have already occurred. Various closure conditions (e.g., under intersection, directed union) are required from $\mathcal{C}$ to comply with the respective semantic interpretation. We can relate an FM $M \in \mathcal{M}(F)$ to a CS $\mathbb{C}(M) = (E, \mathcal{C})$ where $E = F$ and $\mathcal{C} = \mathcal{PP}_M$. Note that $\mathbb{C}(M)$ would be a *connected* CS. A CS is connected if all of its configurations are reachable. A configuration X is reachable if $\exists e_1, \ldots, e_n$ such that $X = \{e_1, \ldots, e_n\}$ and $\forall i \leq n : \{e_1, \ldots, e_i\}$ is also a configuration.

CSs can be specified either directly as above, or indirectly by adding some additional structure to set E. For example, a prime event structure is a triple $\mathbb{E} = (E, \leq, \#)$ with $\leq$ a partial order on E specifying a binary *causality relation* between events, and $\#$ a binary conflict relation (irreflexive and symmetric)

[3] Note that CSs first was introduced in [29]. The original CSs were required to be non-empty, connected and closed under union.

between events, such that the principles of *finite causes* ($\forall e \in E : \{e' \in E : e' \leq e\}$ is finite) and *conflict heredity* ($\forall e, e', e'' \in E : e \leq e' \wedge e\#e'' \Rightarrow e'\#e''$) hold. A set of events X is called a configuration of $\mathbb{E}$, $X \models \mathbb{E}$, if it is left-closed wrt. causality: $(e{\leq}e' \wedge e'{\in}X) \Rightarrow e'{\in}X$, and is conflict-free: $(e\#e' \wedge e{\in}X) \Rightarrow e'{\notin}X$. In a more general ES, called *follow event structure*, the casualty relation does not need to be transitive nor acyclic and also the conflict relation is not assumed to be symmetric, which means we may have inconsistent events. Thus in such ESs the notion of a configuration is respectively generalized (In the most general ES, introduced in [30], both relations $\leq$ and $\#$ are defined on the power set of E instead of E). In this way, an ES determines a CS $\mathbb{C}(\mathbb{E}) = \{X \subseteq E\colon\ X \models \mathbb{E}\}$. We may understand $\mathbb{E}$ as a "logic" over E, and the validity relation as "satisfiability". Then, if $\mathbb{C} = \mathbb{C}(\mathbb{E})$ for a given $\mathbb{C}$, we may say that $\mathbb{E}$ "axiomatizes" $\mathbb{C}$.

CSs can also be axiomatized directly with (infinitary) Boolean propositional logic by considering events as atomic propositions ($e{\in}E$ states that the event e is happend); according to [30], this idea is attributed to Pratt [15]. We explained it in detail in Sect. 2.2 for product lines. Indeed, the framework described above makes perfect sense if we rename events as features (or feature occurrences), configurations as products. (Warning: Subfeature relation in FT is opposed to causality in ES, so that p-products are right- rather than left-closed.). FTs with exclusion constraints then appear as sorts of ESs, while full FMs (FTs + CCCs) as ESs enriched with Boolean constraints. Interestingly, in early work FM [17], axiomatization of FMs was developing along the ES lines, but later the Boolean logic view prevailed; in this sense, Batory's paper [3] played the role of Pratt's paper in concurrency modeling.

The dynamic behaviour (computational interpretation) of a CS $(E, \mathcal{C})$ is usually defined by a transition relation between the configurations. There are two common alternatives: *asynchronous interpretation* from Glabbeek based on the assumption that only finitely many events can happen in a finite time [31], and the interpretation of *Chu spaces* from Gupta & Pratt (eKSs/Gates) [15]. In both systems, it is an unavoidable requirement that a transition from X to Y implies $X \subset Y$. For eKSs/Gates, no other restrictions are imposed and transition relation is inclusion relation. For asynchronous interpretation, valid transitions must satisfy an additional condition: for any $Z{\subset}E$, if $X{\subset}Z{\subset}Y$, then $Z{\in}\mathcal{C}$. It says that "a number of events can be performed concurrently only if they can be performed in any order" [31].

EAs give an asynchronous interpretation of concurrent systems. An EA is a 4-tuple $\mathbb{E} = (E, St, \longrightarrow, I)$, where E is a set of events, $St \subseteq Fin(E)$ a set of states ($Fin(E)$ denotes the finite subsets of E), $\longrightarrow$ a transition relation with $(S \longrightarrow S') \Rightarrow (S' = S \uplus \{e\})$ for some $e \in E$, and I is an initial state. It is proved in [20] that EAs subsume other event-based structures such as prime ESs and follow ESs. An even richer notion is an EA with *quiescent* states [20], and we come to our fKS. In fact, an fKS is syntactic exactly an EA with quiescent states (= full products or self-looped states), which, additionally, satisfies two conditions: (DAG) and (coReach). Note that any EKS can be seen as an EA but not vice versa.

Now we discuss essential distinctions between the two frameworks. In the feature modeling context, we are especially interested in ways of defining EAs syntactically, and in a logic that would allow us to specify EA's properties. Pinna and Poigné [20] only consider a very simple "event logic". In contrast, FMs provide a compact notation for a very rich event logic, in which even modal properties of EAs are implicitly encoded. We can increase expressiveness even more by adding to FMs modal cross-cutting formulas.

Importantly, fKSs generated by FMs, i.e., PPLs, implicitly carry a richer structure inherited from FTs, to wit: a PPL has the notion of an accomplished branch (sub-FKS), and satisfies the **I2C**-constraint. Pinna and Poigné [20] briefly mention such types of constraints as transactions.

5.3 Staged Configuration

Czarnecki et al. [9] introduced the concept of *staged configuration* in which the process of specifying a product is performed in stages, such that in each stage a *specialization* of the given FM is generated by making some choices defined in the FM. This process, called *product derivation process* (PDP), is continued until a fully specialized feature diagram denoting only one configuration is obtained.

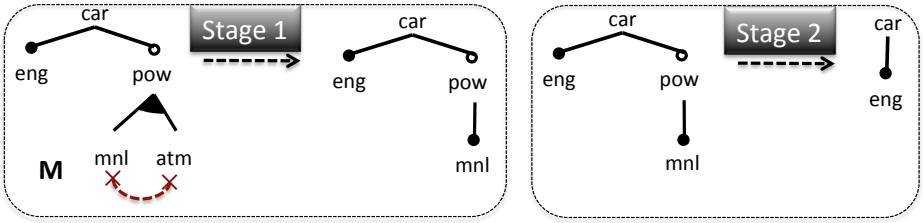


Fig. 11: Staged Configuration

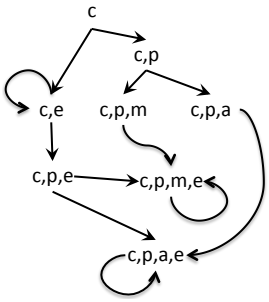


Fig. 12: PPL

Fig. 11 shows an FM M (pow stands for power locks) and a PDP for the full product {car, eng} of this model. It includes two sages: in the first stage, the choice between manual and automated power locks is made and in the second stage, the power locks is eliminated. The corresponding sequence of FMs is called a configuration path [6].

Fig. 12 shows the PPL of M. Let us call a path from the initial state to a full product *assembling path*. Although both assembling and configuration paths show how to instantiate the full product, the difference between their approaches are very profound. Firstly, configuration paths are sequences of FMs, while assembling paths are sequences of partial products. Secondly, in staged configuration, one can make

choices regardless to any condition other than exclusive constraints, while an assembling path shows how to reach the full product by (dis)assembling the features step by step in a top-down fashion considering also the **I2C**-principle. For example, take a look at Fig. 11: in the first stage, we made choices between mnl and atm before making decision whether the car is equipped with a power lockers or not (seems so strange!). Such a decision is not allowed in PPLs: we cannot include a feature into a product without including its parent.

5.4 Algebraic Modeling of Feature Modeling

Höfner et al. developed an algebra, called *product family algebra*, for product lines whose basis is the structure of idempotent semirings [16]. A product family algebra over a set of features F is 5-tuple $\mathcal{A} = (A, +, \varnothing, \times, \{\varnothing\})$ where $A = 2^{2^F}$ (power set of power set of features), $\varnothing$ represents the empty product line, $\{\varnothing\}$ is a dummy/pseudo product line with only one product: nothing, and $+, \times$ are defined as follows: for all $P, P' \in A : P \times P' = \{p \cup p' : p \in P, p' \in P'\}$ and $P + P' = P \cup P'$. In this way, $+$ and $\times$ can be seen as choice between product lines and their mandatory presence, respectively. It is proven that $\mathcal{A}$ forms a semiring where $(A, +, 0)$ and $(A, \times, 1)$ are the commutative monoid and monoid parts, respectively, such that $+$ is idempotent and $\times$ is commutative. Therefore, a product line is seen as a term generated in this algebra.

The product line of a given FM M is encoded as a term in the product line algebra generated over the *basic features* of M. Basic features are those that are labeled to leaves in the feature diagram and this is an important practice in this work, which is in contrast with a common feature modeling practice. Note that we follow the common practice, in which all features in the tree are considered as the basic features. As an example, consider the following feature diagram, which is adopted from [16]. The encoded term corresponding to this FM is as follows: $car = (manual + automatic) \times horsepower \times (1 + aircondition)$.

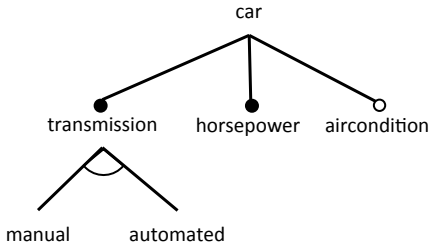


Fig. 13: an FM - adopted from [16] (page 7 , Fig. 1)

To find a precise relation to semirings, we need to algebraicize our approach along the usual lines of algebraic logic — we must leave this for future work. We believe that using KS and modal logic is simpler and easier for a PL engineer than dealing with abstract semiring algebra.

A series of algebraic models for FD and PL has been developed in the context of FOSD [2,11]. Their setting is much more concrete than ours: features are

blocks of code, or other components, whose composition makes a product. This work focuses on the operation of feature/delta composition and delta management.

5.5 Use of FM in behavior modeling.

In a known paper [5], Classen et al. study model checking of a family of transition systems. Such a family is modeled by what they call a *Feature Transition System*, which describes the combined behavior of the entire system. Thus, they consider a PL of behavior models (features are transition systems), whereas we study the behavior pertinent to any PL irrespective of what features are. Applying their technique to our FKS semantics for FD would result in some sort of meta-Kripke structures, which seems to be an interesting object of research.

6 Conclusion

We have presented a dynamic view of feature modeling, in which a product configuration, or instantiation, is seen as a process progressing from partial to full products. Three basic constraints regulating this process are (a) closedness under superfeatures, (b) instantiate-to-completion, and (c) respecting feature mutual exclusion declared in the model. We have shown that PLs encompassing partial products satisfying the constraints are Kripke structures of a special kind that we called *feature KS*. We have also demonstrated that properties of such KSs, and hence PPLs, can be described by a suitable version of modal logic, namely, a fragment of CTL enriched with a constant modality designating full products (final states). We call this logic *feature CTL*. These results establish close connections between feature modeling and modal logic, which we believe can be fruitful for both fields.

References

1. Apel, S., C. Kästner and C. Lengauer, *Featurehouse: Language-independent, automated software composition*, in: *ICSE*, 2009, pp. 221–231.
2. Apel, S., C. Lengauer, B. Möller and C. Kästner, *An algebraic foundation for automatic feature-based program synthesis*, Sci. Comput. Program. **75** (2010), pp. 1022–1047.
3. Batory, D., "Feature models, grammars, and propositional formulas," Springer, 2005.
4. Benavides, D., S. Segura and A. Ruiz-Cortés, *Automated analysis of feature models 20 years later: A literature review*, Information Systems **35** (2010), pp. 615–636.
5. Classen, A., P. Heymans, P.-Y. Schobbens, A. Legay and J.-F. Raskin, *Model checking lots of systems: efficient verification of temporal properties in software product lines*, in: *32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, ACM, 2010, pp. 335–344.
6. Classen, A., A. Hubaux and P. Heymans, *A formal semantics for multi-level staged configuration.*, VaMoS **9** (2009), pp. 51–60.

7. Clements, P. C. and L. Northrop, "Software Product Lines: Practices and Patterns," SEI Series in Software Engineering, Addison-Wesley, 2001.
8. Czarnecki, K. and U. W. Eisenecker, "Generative programming: methods, tools, and applications," ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.
9. Czarnecki, K., S. Helsen and U. Eisenecker, *Staged configuration using feature models*, in: *Software Product Lines*, Springer, 2004 pp. 266–283.
10. Czarnecki, K., S. Helsen and U. Eisenecker, *Formalizing cardinality-based feature models and their specialization*, Software Process: Improvement and Practice **10** (2005), pp. 7–29.
11. Damiani, F., L. Padovani and I. Schaefer, *A formal foundation for dynamic delta-oriented software product lines*, in: K. Ostermann and W. Binder, editors, *GPCE* (2012), pp. 1–10.
12. Eriksson, M., J. Börstler and K. Borg, *The pluss approach: domain modeling with features, use cases and use case realizations*, in: *SPLC 2005*, SPLC'05 (2005), pp. 33–44.
URL `http://dx.doi.org/10.1007/11554844_5`
13. Gheyi, R., T. Massoni and P. Borba, *Automatically checking feature model refactorings*, J. UCS **17** (2011), pp. 684–711.
14. Griss, M. L., J. Favaro and M. d'Alessandro, *Integrating feature modeling with the rseb*, in: *Software Reuse, 1998. Proceedings. Fifth International Conference on*, IEEE, 1998, pp. 76–85.
15. Gupta, V. and V. R. Pratt, *Gages accept concurrent behavior*, in: *FOCS*, 1993, pp. 62–71.
16. Höfner, P., R. Khédri and B. Möller, *An algebra of product families*, Software and System Modeling **10** (2011), pp. 161–182.
17. Kang, K. C., 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).
18. Kang, K. C., S. Kim, J. Lee, K. Kim, E. Shin and M. Huh, *Form a feature oriented reuse method with domain-specific reference architectures*, Annals of Software Engineering **5** (1998), pp. 143–168.
19. Mannion, M., *Using first-order logic for product line model validation*, in: *Software Product Lines*, Springer, 2002 pp. 176–187.
20. Pinna, G. M. and A. Poigné, *On the nature of events: Another perspective in concurrency*, Theor. Comput. Sci. **138** (1995), pp. 425–454.
21. Pohl, K., G. Böckle and F. Van Der Linden, "Software product line engineering: foundations, principles, and techniques," Springer, 2005.
22. Pratt, V., *Event spaces and their linear logic*, in: *AMAST'91* (1992).
23. Riebisch, M., K. Böllert, D. Streitferdt and I. Philippow, *Extending feature diagrams with uml multiplicities*, , **50**, Citeseer, 2002.
24. Roos-Frantz, F., D. Benavides and A. Ruiz-Cortés, *Feature model to orthogonal variability model transformation towards interoperability between tools*, KISS@. ASE, New Zealand (2009).
25. Safilian, A., T. Maibaum and Z. Diskin, *The semantics of feature models via formal languages: a computational hierarchy for feature models*, Technical Report; TR 2014-08-01, McMaster University (CAS), McMaster (2014).
26. Schobbens, P.-Y., P. Heymans, J.-C. Trigaux and Y. Bontemps, *Generic semantics of feature diagrams*, Computer Networks **51** (2007), pp. 456–479.
27. She, S., R. Lotufo, T. Berger, A. Wasowski and K. Czarnecki, *Reverse engineering feature models*, in: *ICSE 2011*, IEEE, 2011, pp. 461–470.

28. Trujillo, S., D. S. Batory and O. Díaz, *Feature oriented model driven development: A case study for portlets*, in: *ICSE*, 2007, pp. 44–53.
29. van Glabbeek, R. and U. Goltz, *Refinement of actions in causality based models*, in: *Stepwise Refinement of Distributed Systems Models, Formalisms, Correctness*, Springer, 1990, pp. 267–300.
30. van Glabbeek, R. J. and G. D. Plotkin, *Configuration structures*, in: *LICS*, 1995, pp. 199–209.
31. van Glabbeek, R. J. and G. D. Plotkin, *Configuration structures, event structures and petri nets*, Theoretical Computer Science **410** (2009), pp. 4111–4159.
32. Van Gurp, J., J. Bosch and M. Svahnberg, *On the notion of variability in software product lines*, in: *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*, IEEE, 2001, pp. 45–54.
33. Winskel, G., *An introduction to event structures*, in: J. W. de Bakker, W. P. de Roever and G. Rozenberg, editors, *REX Workshop*, Lecture Notes in Computer Science **354** (1988), pp. 364–397.