GSDLAB TECHNICAL REPORT

Modeling Product Lines with Kripke Structures and Modal Logic

Zinovy Diskin, Aliakbar Safilian, Tom Maibaum

GSDLAB–TR 2013-10-1 October 2013

Generative Software
Development Lab

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

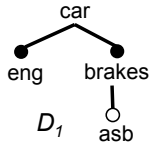
Zinovy Diskin, Aliakbar Safilian, Tom Maibaum

Generative Software Development Lab.,
Department of Electrical and Computer Engineering,
University of Waterloo, Canada
zdiskin@gsd.uwaterloo.ca

Abstract. Product lines (PLs) is a well known framework for software design, which has recently been gaining considerable attention in research and industry. PLs are defined by so called feature models (FMs): graphical structures showing products' features and relationships between them. A common way for defining a formal semantics for FMs is to treat them as propositional formulas with the ordinary Boolean semantics. A major drawback of this setting is that intuitively very different FMs can have the same Boolean semantics, and thus important information is lost when FMs are translated into propositional formulas. In the paper we show how this problem can be fixed in the framework of modal logic and its Kripke semantics. We introduce *feature Kripke structures* as a semantic universe for PLs, and *feature modal logic* as a syntactic universe for FMs, and reformulate basic notions of feature modeling in this refined setting. Moreover, the framework provides richer expressiveness for describing FMs: new useful properties of PLs can be specified and verified.

1 Introduction

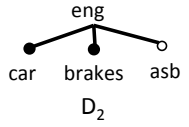
The *Software Product Line* approach is well-known in the software industry. Products in a product line (PL) share some set of common features, and differ by having some variable features that allow the user to configure the product the user wants. Thus, instead of producing a multitude of separate products, a single PL encompassing the variety of products is designed, which results in a significant reduction in development time and cost [16].



PLs are defined by so called feature models (FMs) based on feature diagrams (FDs): graphical structures showing products' features and relationships between them. The inset figure presents a simple example. The FD D_1 says that each car has two *mandatory* features (as indicated by black circles), eng and brakes, and the latter has an *optional* feature (hollow circle) asb. This defines a PL with two products, i.e., two legitimate sets of features: $P_1 = \{\mathsf{car}, \mathsf{eng}, \mathsf{brakes}\}$ and $P_2 = P1 \cup \{\mathsf{asb}\}$. Industrial FDs may have thousands of features, and their PLs can be quite complex [15]. To manage their design and analysis, PLs and

FDs should be represented as formal objects processable by tools. A common approach is to consider features as propositional variables, and encode FDs by formulas of Boolean propositional logic (**BL**), whose valid valuations are to be valid products defined by the FD [2]. This approach gave rise to a series of prominent applications to analysis of industrial size PLs [3, 10, 18].

However, the **BL** encoding has an inherited drawback. The inset figure shows another FD specifying an entirely different product line (in fact, pathological), but this FD has the same set of products $\{P_1, P_2\}$ as the earlier FD. Obviously, some important aspects of a FD's semantics are not captured by their Boolean semantics and lost with their **BL**-translation (this lost semantics was called *ontological* in [18]). This deficiency of BL is known, but as far as we know, no logic has been proposed to replace **BL** to fix the problem.



In the paper we show that what is lost in the **BL**-encoding is the *dynamic* nature of the notion of product. An FD defines not just a set of valid products but the very way these products are to be assembled step by step from constituent features. Correspondingly, a PL appears as a transition system initialized at the root feature (say, car) and gradually progressing towards fuller products (say, $\{\mathsf{car}\} \to \{\mathsf{car}, \mathsf{eng}\} \to \{\mathsf{car}, \mathsf{eng}, \mathsf{brakes}\}$ or $\{\mathsf{car}\} \to \{\mathsf{car}, \mathsf{brakes}\} \to \{\mathsf{car}, \mathsf{brakes}, \mathsf{asb}\} \to \{\mathsf{car}, \mathsf{brakes}, \mathsf{asb}, \mathsf{eng}\}$. Moreover, we will show that relations between products are richer than merely inclusions: while any transition is a set inclusion, not any inclusion is a legal transition. In this way, FDs are provided with a Kripke rather than Boolean semantics, and their logical counterparts are modal rather than Boolean formulas. *Product lines are Kripke structures* is the main motto of the present paper.

Our plan is as follows. In the next section we recall the basic definitions of feature modeling, and formalize them in a convenient way for our purposes. In Section 3 we discuss the dynamic aspects of a FD's semantics, develop a framework of basic concepts, and prove several initial results. In particular, we motivate and formalize an important semantic requirement called *instantiate-to-completion* (so called by analogy with the "run-to-completion" semantics known in behavior modeling). In Section 4, we introduce *feature Kripke structures* and *feature Computation Tree logic* (**fCTL**) — simple specializations of Kripke structures and the logic CTL known from behavior modeling and model checking, but still having many non-trivial properties. We also present an example showing how **fCTL** can be used for specifying non-trivial properties of PLs beyond **BL**. Related work is discussed in Section 5, and Section 6 concludes.

2 Background: Feature Diagrams and Models

There are many feature modelling languages. Some of them, including the classical FODA [13], are tree-based, e.g., [5, 9], others are DAG-based such as [14, 17]. In this paper, we restrict our attention to feature trees, and do not consider cardinality-based feature modeling [7], in which the same feature can occur in the product multiple times.

In a typical FM notation, non-root features are either *solitary* or *grouped*. Solitary features are either mandatory (e.g., eng, trans, and brakes in Fig. 1) or optional (like asb). Feature groups usually include OR-groups (inclusive OR, e.g., eng can be either gas or elec, or both in Fig. 1), and XOR-gorups (exclusive OR, trans is either mnl or atm but not both). Less common but useful is the MEX-group (mutual exclusion), which admits at most one of mutually exclusive features occurs into the product. FODA-like notations for these constructs are summarized in Table 1 (ignore the third row for a while), and a formal definition is given by Def. 1.

Definition 1 (Feature Diagrams). A *feature diagram* (FD) is a triple $\mathbb{D} = (\mathbb{F}, \mathcal{S}, \mathcal{G})$ of the following components.

(i) $\mathbb{F} = (r, F, \_^{\uparrow})$ is a *feature tree* with feature r as the root, set F of non-root features, and function $\_^{\uparrow}: F \to F \cup \{r\}$ that maps each non-root feature f to its parent $f^{\uparrow}$. The inverse function that assigns to each feature the set of its children is denoted by $\_{\downarrow}$; this set is empty for leaf features. The set of grandparents and grandchildren of a feature f are denoted by $f^{\uparrow\uparrow}$ and $f_{\downarrow\downarrow}$ resp.

(ii) $\mathcal{S} = (M \uplus O) \subseteq F$ is a set of *solitary* features partitioned into *mandatory* (set M) and *optional* (set O) features.

(iii) For each predicate $p \in \{or, xor, mex\}$, a disjoint family $\mathcal{G}_p \subseteq 2^{(F \setminus \mathcal{S})}$ of p-labeled sets (called *groups*) of non-solitary features is given, such that if $G \in \mathcal{G}_p$, then cardinality $|G| > 1$ and all features in G have the same parent, denoted by $G^{\uparrow}$. The three families are themselves disjoint and together cover all non-solitary features: any non-root feature f is either solitary, $f \in \mathcal{S}$, or belongs to one and only one group, $f \in G \in \mathcal{G} = \mathcal{G}_{xor} \uplus \mathcal{G}_{or} \uplus \mathcal{G}_{mex}$ (and then we say f is a *grouped* feature). □

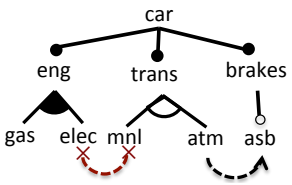


Fig. 1: A sample feature model.

Remark 1 (On notation). Below we will try to follow the notational discipline used above. Sets are typed in italic with upper case letters, like X, their elements are lower case, like $x \in X$, and sets of sets are calligraphic, like $X \in \mathcal{X}$. Sets equipped with additional structure are in "mathbold" e.g., a poset $(X, \preceq)$, or a set of sets ordered by inclusion, $(\mathcal{X}, \subseteq)$, would be denoted by $\mathbb{X}$. We will allow ourselves some deviations, e.g., F is not the entire carrier set of tree $\mathbb{F}$, if they simplify technicalities, or make reading easier.

Any feature diagram $\mathbb{D} = (\mathbb{F}, \mathcal{S}, \mathcal{G})$ can be translated into a propositional formula $\Phi(\mathbb{D})$ [2]. Features are considered as propositional variables, and basic FD constructs are encoded as shown in Table 1. Each encoding is a formula or a conjunction of formulas; the latter are than shown in different rows. The upper row encodes the subfeature relationship present in every construct. The second row is for an additional *mandatoriness* requirement specified by the construct (there may be none such), and the third row encodes MEX-requirements (again optional). Finally, we add formula $\top \Leftrightarrow r$ for the root feature.

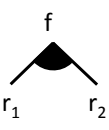
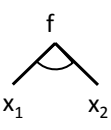
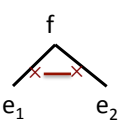
Subfeature	Mandatory feature m	Optional feature o	OR group	XOR group	MEX group
					
$f \Leftarrow f'$	$f \Leftarrow m$ $f \Rightarrow m$	$f \Leftarrow o$	$f \Leftarrow r_1 \vee r_2$ $f \Rightarrow r_1 \vee r_2$	$f \Leftarrow x_1 \vee x_2$ $f \Rightarrow x_1 \vee x_2$ $x_1 \wedge x_2 \Rightarrow \bot$	$f \Leftarrow e_1 \vee e_2$ $e_1 \wedge e_2 \Rightarrow \bot$

Table 1: Translating FDs to **BL**

A *feature model* is an FD with a set of additional *cross-cutting* constraints (we will write CCCs) for features in different branches of the feature tree. Typically, such constraints are either *exclusive* or *inclusive*. For example, the $\times$-ended arc in Fig. 1 denotes an exclusive CCC that prohibits having both an electric engine and a manual transmission in a product. The arrow denotes an *inclusive* CCC that requires a car with automatic transmission to also have asb. In general, an inclusive constraint has the format $f_1 \wedge \ldots \wedge f_n ..... \rightarrow f$, and an exclusive one is $f_1 \wedge \ldots \wedge f_n ..... \rightarrow \neg f$ with f_i features in any positions in the feature tree. Note that an exclusive constraint is equivalent to the following mex-constraint $f_1 \wedge \ldots \wedge f_n \wedge f \rightarrow \bot$, which means that, taken together, the $(n+1)$ features are incompatible (but any proper subset of them may still be compatible!).

Definition 2 (Feature Models). A *feature model* (FM) is a pair $\mathbb{M} = (\mathbb{D}, \Phi_{ccc})$ with $\mathbb{D}$ an FD and Φ_{ccc} a conjunction of exclusive and inclusive constraints built over F interpreted as a set of propositional variables. □

We will often write an FM as a triple $(\mathbb{D}, \Phi_{in}, \Phi_{ex})$ with Φ_{in} being a conjunction of inclusive, and Φ_{ex} of exclusive, CCCs. Thus, an FM is basically a triple of propositional formulas $\mathbb{M} = (\Phi_{\mathbb{D}}, \Phi_{in}, \Phi_{ex})$ with $\Phi_{\mathbb{D}}$ denoting $\Phi(\mathbb{D})$ as specified above.

Definition 3 (Full products). A *full product* over $\mathbb{D} = (\mathbb{F}, \mathcal{S}, \mathcal{G})$ is a set of features $P \subseteq F \cup \{r\}$ such that the formula $\Phi(\mathbb{D})$ is satisfied in the Boolean sense: if all variables $f \in P$ are evaluated to true, then the term $\Phi(\mathbb{D})$ is evaluated to true as well. We then write $P \models_{\mathbf{BL}} \mathbb{D}$. The set of all full products is called a *full product line* over $\mathbb{D}$ and denoted by $\mathcal{FP}(\mathbb{D})$ (while the entire powerset is denoted by 2^F). Thus, $\mathcal{FP}(\mathbb{D}) = \{P \subseteq F\colon\ P \models_{\mathbf{BL}} \mathbb{D}\}$.

A *(full) product* over $\mathbb{M} = (\mathbb{D}, \Phi_{ccc})$ is a product over $\mathbb{D}$, which, in addition, satisfies all constraints in Φ_{ccc}. This gives rise to the product line $\mathcal{FP}(\mathbb{M}) = \{P \subseteq F\colon\ P \models_{\mathbf{BL}} \mathbb{D} \text{ and } P \models_{\mathbf{BL}} \Phi_{ccc}\}$. □

Remark 2. The definition above is standard, except that we use the term *full* product rather than product. The reason is that later we will also introduce *partial* products. We will often use abbreviations f-product and p-product for full and partial products, resp.

3 Partial Product Lines: Instantiation as a Process

Let $\mathbb{D} = (\mathbb{F}, \mathcal{S}, \mathcal{G})$ be an FD as defined in Section 2. Our main observation is that for finding its proper semantics, we need to take into account not only full products over $\mathbb{D}$, but the very way these products are assembled (we will say *instantiated*) starting from the root feature and then being gradually augmented with new features until leaves in tree $\mathbb{F}$ are reached; full products assembled in this way are specially marked. The instantiation process is regulated by the FD's extra structure $\mathcal{S}$ and $\mathcal{G}$, which actually specifies a set of constraints for product assembly. The goal of this section is to analyse this process in detail, highlight its main points, and accurately formalize it.

3.1 Getting started

Figure 2 shows how basic FD constructs guide the process of product instantiation. The first row presents six basic FD constructs, whereas the second row

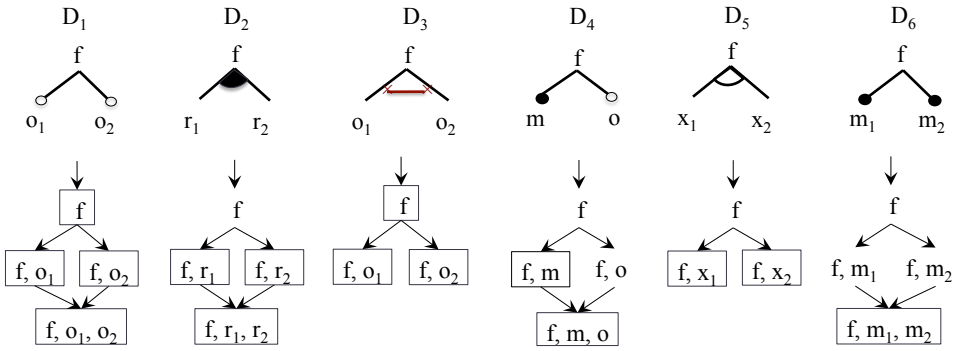


Fig. 2: From FDs to PPLs. The boxed states denote full products.

shows their meaning in terms of top-down product design as *instantiation diagrams*. Nodes of instantiation diagrams denote *partial* products (*p-products*), i.e., legal products with, perhaps, some features missing, for example, product {f, o} in diagram $\mathbb{D}_4$ is missing the feature m. Edges are inclusions between p-products; in fact, each edge encodes adding a single feature to the product at the source of the edge. p-products that are full products (f-products) are boxed. Thus, in contrast to the ordinary Boolean semantics of feature diagrams, which merely lists f-products, instantiation diagrams specify how f-products are gradually instantiated step by step. This focus on the process rather than the result is typical for Kripke semantics, and indeed, in Sect. 4 we will show that our instantiation diagrams are nothing but a special case of Kripke structures. We will also call instantiation diagrams *partial product lines* (PPLs).

Fig. 3 presents instantiation diagrams for the two FDs considered in the Introduction (c, e, b, and a stand for car, eng, brakes, and asb). While both

FDs have the same set of f-products (i.e., are Boolean semantics equivalent), their PPLs (Kripke semantics) are essentially different and properly capture the difference of the FDs.

Now we consider how instantiation diagrams are built in more detail. We will begin in the next section with an auxiliary but useful approximation (called *free* p-product) of what a legal p-product should be. Then, in Sect. 3.3, we discuss an important principle of product assembly, *instantiate-to-completion*, which we believe is pertinent to FM although is implicit in FDs. We named it so because the principle is analogous to the well-known "run-to-completion" semantics in behavior modeling.

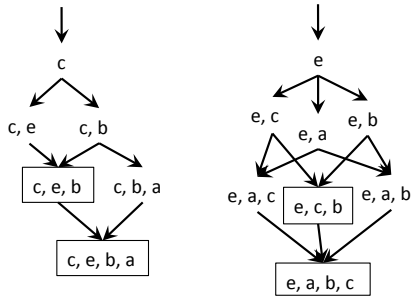


Fig. 3: PPLs of FDs in the Introduction.

3.2 Free partial products

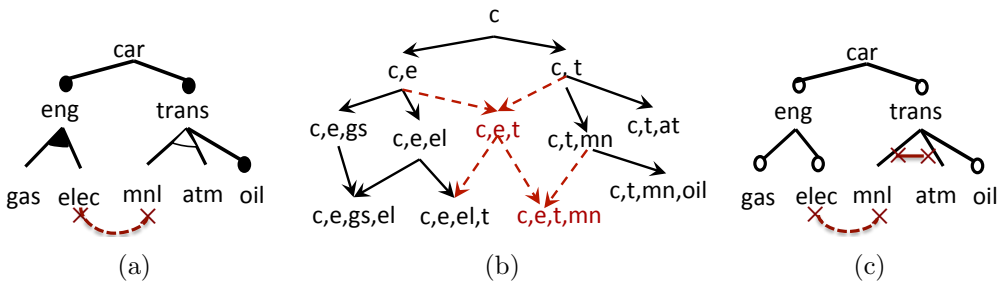


Fig. 4: Free PPLs and **I2C** constraint

Figure 4(a) shows an FD, and Figure 4(b) describes its partial products understood as free combinations of basic features: c, e, t stand for upper-level features car, eng, trans, and two-letter abbreviations gs, el, mn, and at are for their subfeatures. (Ignore for a while that the node {c, e, t} is typed in grey (red with a color display), and arrows adjoint to it are grey and dashed.) The idea is that any combination is possible unless it is prohibited by an exclusive constraint declared either in a MEX- or XOR-group, or in Φ_{ex}. For example, the node combining the features gs and el does exist, whereas combinations {mn, at} and {el, mn} are prohibited. Mandatoriness of solitary features, and mandatoriness embodied into XOR- and OR-groups, do not affect free product generation (but do determine what full products are). In fact, a free p-product over an FD $\mathbb{D}$ is a full product over a relaxed FD $\mathbb{D}^{\circ}$ with all mandatory requirements stripped. For example, Figure 4(c) presents such a relaxation of FD Figure 4(a). The definition below makes this construction precise.

Definition 4 (Free Partial Products). A *free p-product* over an FD $\mathbb{D} = (\mathbb{F}, \mathcal{S}, \mathcal{G})$ is a full product over the FD $\mathbb{D}^\circ = (\mathbb{F}^\circ, \mathcal{S}^\circ, \mathcal{G}^\circ)$, with $\mathbb{F}^\circ = \mathbb{F}$, $M^\circ = \varnothing$, $O^\circ = O \cup M \cup (\bigcup \mathcal{G}_{or})$, and ${\mathcal{G}_{mex}}^\circ = \mathcal{G}_{mex} \cup \mathcal{G}_{xor}$, ${\mathcal{G}_{or}}^\circ = {\mathcal{G}_{xor}}^\circ = \varnothing$.

A *free p-product* over an FM $\mathbb{M} = (\mathbb{D}, \Phi_{in}, \Phi_{ex})$ is a free p-product over $\mathbb{D}$, which additionally satisfies the constraints in Φ_{ex}. In other words, it is a full product over $\mathbb{M}^\circ = (\mathbb{D}^\circ, \Phi^\circ_{ex})$.

We denote the set of all free p-products over $\mathbb{D}$ and $\mathbb{M}$ by $\mathcal{PP}_{\text{free}}(\mathbb{D})$ and $\mathcal{PP}_{\text{free}}(\mathbb{M})$, respectively. □

The definition of free products admit a useful equivalent reformulation.

Proposition 1. Let $\mathbb{D} = (\mathbb{F}, \mathcal{S}, \mathcal{G})$ be an FD. A set of features $P \subseteq F \cup \{r\}$ is a free p-product iff it satisfies the following two conditions: (a) if $f \in P$ then $f^\uparrow \in P$ as well (we say that P is *up-closed*), (b) if a group of features $G \in \mathcal{G}_{mex} \cup \mathcal{G}_{xor}$, then $|P \cap G| \leq 1$.

A set of features over an FM $\mathbb{M} = (\mathbb{D}, \Phi_{in}, \Phi_{ex})$ is a free p-product iff it satisfies conditions (a), (b) above, and condition (c) $P \models_{\mathbf{BL}} \Phi_{ex}$. □

Proof. According to Definition 4, a free p-product of $\mathbb{M}$ is a f-product of $\mathbb{M}^\circ$. Consider a full product P° of $\mathbb{M}^\circ$. Since $\mathbb{F}^\circ = \mathbb{F}$ and a f-product of an FM is up-closed, $\forall f \in P^\circ$. $f^\uparrow \in P^\circ$. Since $\mathcal{G}^\circ_{mex} = \mathcal{G}_{mex} \cup \mathcal{G}_{xor}$ and $\Phi^\circ_{ex} = \Phi_{ex}$, the conditions (b) and (c) follow obviously.

Now, consider a set of features P which satisfies the conditions (a), (b), and (c). We want to prove that P is a f-product of $\mathbb{M}^\circ$, i.e., $P \models_{BS} \Phi^\circ_{\mathbb{D}} \wedge \Phi^\circ_{ex} \wedge \Phi^\circ_{in}$. To prove $P \models_{BS} \Phi^\circ_{\mathbb{D}}$, we need just to show that P satisfies $\mathcal{G}^\circ_{mex}$, ($M^\circ = \mathcal{G}^\circ_{or} = \mathcal{G}^\circ_{xor} = \varnothing$). Since $\mathcal{G}^\circ_{mex} = \mathcal{G}_{mex} \cup \mathcal{G}_{xor}$, $P \models_{BS} \mathcal{G}^\circ_{mex}$ follows obviously by the condition (b). Since $\Phi^\circ_{ex} = \Phi_{ex}$, (c) implies directly that $P \models_{BS} \Phi^\circ_{ex}$. Finally, $P \models_{BS} \Phi^\circ_{in}$ because $\Phi^\circ_{in} = \top$. □

Proposition 2. For a given FD, its set of free p-products is closed under intersection. The same holds for the set of free p-products for an FM. □

Proof. Consider two free partial products P and P' of an FM $\mathbb{M}$. Since, according to Proposition 1, P and P' are up-closed, $\forall f \in P \cap P'$. $f^\uparrow \in P \cap P'$ and so $P \cap P'$ is up-closed $(*)$.

Now, consider a group of features $G \in \mathcal{G}_{mex} \cup \mathcal{G}_{xor}$. Since, according to Proposition 1, $|P \cap G| \leq 1$ and $|P' \cap G| \leq 1$, it follows obviously that $|P \cap P' \cap G| \leq 1$ $(**)$.

According to Proposition 1, $P \models_{BS} \Phi_{ex}$ and $P \models_{BS} \Phi_{ex}$, which mean for any conjunctive clause $c_{ex} : f_1 \wedge \cdots \wedge f_n \Rightarrow \neg f'$ in Φ_{ex} (for features $f', f_1, \cdots, f_n$), the satisfactions $\bigwedge_{f \in P} f \models_{BS} c_{ex}$ (1) and $\bigwedge_{f \in P'} f \models_{BS} c_{ex}$ (2) hold. Assume that $\bigwedge_{f \in P \cap P'} f \models f_1 \wedge \cdots \wedge f_n$, which means $\{f_1, \cdots, f_n\} \subseteq P \cap P'$ (3). (1), (2), and (3) imply $P \cap P' \models_{BS} \neg f'$. This leads us to $P \models_{BS} \Phi_{ex}$ $(***)$.

From $(*)$, $(**)$, and $(***)$, according to Proposition 1, the set of free partial products are closed under intersection. □

Thus, free p-products respect subfeature relationships and exclusive constraints, and do not care about anything else. In the next section, we will see that a

majority of free p-products ignore an important instantiate-to-completion requirement, and hence should be discarded as (although algebraically possible, but practically) not interesting.

3.3 Instantiate-to-Completion: Motivation

Instantiation diagrams present a branching-time view of the product assembly. The *instantiate-to-completion* principle (abbreviated by **I2C**) dictates that we begin processing a new branch only after processing of the previous one is completed. For example, a transition from product $\{\mathsf{c}, \mathsf{e}\}$ to product $P = \{\mathsf{c}, \mathsf{e}, \mathsf{t}\}$ violates **I2C** because the feature t is added before the feature e was fully assembled/configured: indeed, we still need to decide whether the e (engine) is to be el (electric) or gs (gasoline). The other transition to the same product, $\{\mathsf{c}, \mathsf{t}\} \longrightarrow P$, also violates **I2C**: now feature e is added before the choice between mn (manual) and at (automatic) transmission is made. Hence, these transitions should be excluded from the instantiation diagram (we make them dashed/red). Now product P becomes unreachable and must be also excluded, together with two outgoing transitions. Note that transition $\{\mathsf{c}, \mathsf{e}, \mathsf{el}\} \longrightarrow \{\mathsf{c}, \mathsf{e}, \mathsf{el}, \mathsf{t}\}$, which begins transmission instantiation in the engine branch, complies with **I2C** as its source is a fully instantiated e(ngine). In contrast, transition $\{\mathsf{c}, \mathsf{t}, \mathsf{mn}\} \longrightarrow \{\mathsf{c}, \mathsf{t}, \mathsf{mn}, \mathsf{e}\}$ violates **I2C** as assembling t is not finished yet due to the presence of mandatory feature oil. The node $\{\mathsf{c}, \mathsf{e}, \mathsf{t}, \mathsf{mn}\}$ becomes unreachable and must be removed from the PPL. Thus, for a given $\mathbb{D}$, its PPL of free products satisfying **I2C**, $\mathcal{PP}_{\mathbf{I2C}}(\mathbb{D})$, can be much smaller than the free PPL, $\mathcal{PP}_{\text{free}}(\mathbb{D})$.

From now on, by a p-product we will understand a free p-product satisfying **I2C**, and the subindex **I2C** will be skipped, that is, given $\mathbb{D}$, the set of free p-products satisfying **I2C** is denoted by $\mathcal{PP}(\mathbb{D})$. Below in Sect. 3.4, we will give an accurate formalization of this construction. The main idea is that we consider a feature f to be fully instantiated at a p-product P, if P is a final product for the FD rooted at f and consisting of all f's grandchildren.

There are situations in which the **I2C** constraint keeps all products in $\mathcal{PP}_{\text{free}}(\mathbb{D})$, but removes some of the transitions. Consider FD in Fig. 5 (on the left) and its PPL (right). "Diagonal" transition $\{\mathsf{car}, \mathsf{trans}\} \longrightarrow \{\mathsf{car}, \mathsf{trans}, \mathsf{eng}\}$ 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. An important consequence of this observation is that the PPL of an FD $\mathbb{D}$ is actually a transition system $\mathbb{P} = (\mathcal{PP}(\mathbb{D}), \longrightarrow)$ with the transition relation being an independent component rather than merely being set inclusion (although it is always a subrelation of the subset relation).

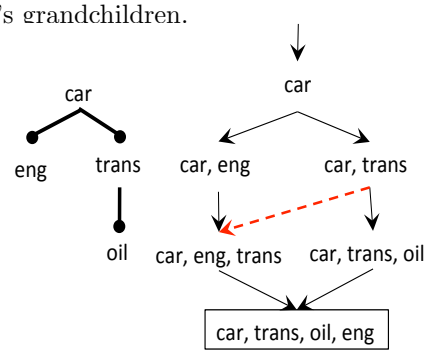


Fig. 5: An FD and its PPL

3.4 Instantiate-to-completion: Formalization

To formalize the notion of p-product satisfying **I2C**, we will first need the following notion.

Definition 5 (Induced Subdiagram). Consider a FD $\mathbb{D} = (\mathbb{F}, \mathcal{S}, \mathcal{G})$ and a feature f. The subdiagram *induced* by f is the following FD $\mathbb{D}^f = (\mathbb{F}^f, \mathcal{S}^f, \mathcal{G}^f)$ where:

(i) $\mathbb{F}^f$ is the tree under f, i.e., the tree $(f, f_{\downarrow\downarrow}, \_^{\uparrow})$

(ii) $\mathcal{S}^f$ is inherited from $\mathcal{S}$, i.e., $O^f = O \cap f_{\downarrow\downarrow}$ and $M^f = M \cap f_{\downarrow\downarrow}$

(iii) $\mathcal{G}^f$ is also inherited from $\mathcal{G}$, i.e., for all predicates $p \in \{or, xor, mex\}$, $\mathcal{G}_p^f = \{G \subseteq f_{\downarrow\downarrow} :\ G \in \mathcal{G}_p\}$. □

Now we can define (**I2C**-compatible) p-products:

Definition 6 (Partial Products). Let $\mathbb{D} = (\mathbb{F}, \mathcal{S}, \mathcal{G})$ be an FD.

A set $P \subseteq F$ is called a *(valid) p-product* if and only if it is (a) a free partial product over $\mathbb{D}$, and (b) the following condition is satisfied:

for any two distinct features with the same parent, $f_1^{\uparrow} = f_2^{\uparrow}$, if both $f_1 \in P$ and $f_2 \in P$, then there exists a set $P^! \subseteq P$ such that it is a full product of the sub diagram induced by f_1 or f_2, i.e., $P^! \in \mathcal{FP}(\mathbb{D}^{f_1}) \cup \mathcal{FP}(\mathbb{D}^{f_2})$.

If $\mathbb{M} = (\mathbb{D}, \Phi_{in}, \Phi_{ex})$ is an FM, then any partial product P over $\mathbb{D}$ is also a *partial product* over $\mathbb{M}$ if, in addition, P satisfies all exclusive constraints, i.e., for all conjunctive clauses ϕ in Φ_{ex}, $P \models_{BS} \phi$.

We denote the set of all partial products for a given FM $\mathbb{M}$ by $\mathcal{PP}(\mathbb{M})$. □

Definition 7 (Extension). Let P, P' are two partial products. We say that P' *extends* P, and write $P \longrightarrow P'$, if $P' = P \cup \{f'\}$ for some feature $f' \notin P$, and there exists $f \in P$ such that

(i) $f = f'^{\uparrow}$ or

(ii) $f^{\uparrow} = f'^{\uparrow}$ and there exists $P'' \subseteq P$ such that $P'' \in \mathcal{FP}(\mathbb{D}^f)$. □

Definition 8 (Partial Product Line). Let $\mathbb{M} = (\mathbb{D}, \Phi_{ccc})$ be an FM.

The set of its partial products equipped with the extension relation, the initial product consisting of the root feature $P_0 = \{r\}$, and the set of its full products, is called *partial product line*, and denoted by $\mathbb{P}(\mathbb{M}) = (\mathcal{PP}(\mathbb{M}), \longrightarrow_{\mathbb{M}}, P_0, \mathcal{FP}(\mathbb{M}))$. □

Importantly for applications, we have developed an algorithm that returns $\mathbb{P}(\mathbb{M})$ for a given feature model $\mathbb{M}$. The algorithm is described as follows:

The main algorithm is ***fm_PPL*:** which returns the PPL assigned to the given FM. The function *Final_States_Finder* outputs the set of final product which can be done by SAT-based tools [2].
The global variables are :

- A given FM $\mathbb{M} = (\mathbb{D}, C_{ex}, C_{in})$ which is considered as the input of fm_PPL.
- The output is $\mathbb{P}(\mathbb{M}) = (\mathcal{PP}(\mathbb{M}), \longrightarrow_{\mathbb{M}}, P_0, \mathcal{FP}(\mathbb{M}))$.
- A subset of features $Q \subseteq F$ which is an axillary variable.

***fm_PPL*:**

1. *Initialization*
2. While $Q \neq \emptyset$ do
   (a) Extract an element $q \in Q$
   (b) Do $All\_Paths\_of\_Subfeatures(q)$
   (c) $Q := Q - \{q\}$
3. Do $Employ\_Exclusion\_Constraints$
4. Do $Final\_States\_Finder$

***Initialization*:**

1. $\mathcal{PP}(\mathbb{M}) := \{r\}$
2. $P_0 := \{r\}$
3. $Q := \{r\}$

***All_Paths_of_Subfeatures(q: F)*:**

1. Take the set $St = \{w \in \mathcal{PP}(\mathbb{M}) \mid q \in w \wedge \forall w' \in \mathcal{PP}(\mathbb{M}).\ (w' \longrightarrow_{\mathbb{M}} w) \Rightarrow q \notin w'\}$
2. Take the set $Subf = \{f \in F \mid q = f^{\uparrow}\}$
3. For all $s \in St$ do
   (a) For all $s' \in \mathcal{PP}(\mathbb{M})$ with $s \longrightarrow_{\mathbb{M}} s'$ do:
      i. For all permutation $seq \in Permutations(Subf)$ do: Add the path $\pi = Create\_Path(seq, s, s')$
4. $Q := Q - \{q\}$

***Create_Path(seq: Sequence of features, s, s': $\mathcal{PP}(\mathbb{M})$)*:**

1. $Q := Q \cup \{f \in seq\}$
2. Define the set of states $\{s_i\}_{1 \leq i \leq |seq|}$
3. $s_1 = s \cup \{seq_1\}$
4. For all $1 \leq i < |seq|$ do: $s_{i+1} = s_i \cup \{seq_{i+1}\}$
5. $\mathcal{PP}(\mathbb{M}) := \mathcal{PP}(\mathbb{M}) \cup \{s_i \mid 1 \leq i \leq |seq|\}$
6. $\longrightarrow_{\mathbb{M}} := \longrightarrow_{\mathbb{M}} \cup \{(s, s_1)\} \cup \{(s_i, s_{i+1}) \mid 1 \leq i < |seq|\} \cup \{(s_{|seq|}, s')\} - \{(s, s')\}$
7. $\longrightarrow_{\mathbb{M}} := \longrightarrow_{\mathbb{M}} \cup \{(s_i, s_i) \mid 1 \leq i \leq |seq|\}$

***Employ_Exclusion_Constraints*:**

1. Take the set $EX = \{A \subseteq F \mid A \subseteq XOR \vee A \subseteq MEX \vee A \subseteq C_{ex}\}$
2. For all $A \in EX$ do:
   (a) For all $s \in \mathcal{PP}(\mathbb{M})$ do:
      i. If $|A \cap s| > 1$ Then $Eliminate\_State(s)$

***Eliminate_State(s: $\mathcal{PP}(\mathbb{M})$)*:**

1. $\mathcal{PP}(\mathbb{M}) := \mathcal{PP}(\mathbb{M}) - \{s' \mid s \longrightarrow^{+} s'\} - \{s\}$
2. $\longrightarrow_{\mathbb{M}} := \longrightarrow_{\mathbb{M}} \cap (\mathcal{PP}(\mathbb{M}) \times \mathcal{PP}(\mathbb{M}))$

4 Feature Kripke Structures and Their Modal Logic

In this section, we adopt the language of CTL (*Computation Tree Logic*) for feature modeling (we call it *Feature CTL*, **fCTL**), and build a semantics for it in terms of *Feature Kripke Structures*. The latter can be seen as an immediate abstraction of PPLs. In Sect. 4.3 we show how the additional expressiveness of **fCTL** can be used for specifying PPL-properties that cannot be captured in the Boolean logic.

4.1 Feature Kripke Structures and feature CTL

A *Kripke structure (KS)* is a loose term denoting a family of mathematical structures of the following format. We first fix a set A of atomic propositions, and then consider a tuple $\mathbb{W} = (W, \longrightarrow, L)$ with W a set of *(possible) worlds*, $\longrightarrow$ a binary *transition* relation over W, and L a labeling function $W \to 2^A$, which maps a world to the set of propositions true in this world. Product lines motivate a specialization of the notion, in which worlds (called products) are identified with sets of atomic propositions (features), and hence labeling is not needed.

Definition 9 (Feature Kripke Structure (FKS)). Let F be a finite set (of *features*). An *FKS* over F is a tuple $\mathbb{P} = (\mathcal{PP}, P_0, \longrightarrow, \mathcal{FP})$ with $\mathcal{PP} \subset 2^F$ a set of *(partial) products*, $P_0 \in \mathcal{PP}$ the *initial* singleton product (its only member is called the *root feature* of the FKS), and $\longrightarrow \subseteq \mathcal{PP} \times \mathcal{PP}$ a *transition* relation, and $\mathcal{FP} \subseteq \mathcal{PP}$ a subset of *full* products.

The following *DAG*, *Singletonity*, and *Finality* conditions are to hold.

(DAG) $(\mathcal{PP}, P_0, \longrightarrow)$ is a rooted DAG with the root P_0
(Sing) if $P \longrightarrow P'$ then $P \subset_f P'$, which means $P' = P \cup \{f\}$ and $f \notin P$
(Fin) for any $P \in \mathcal{PP}$ there is $P' \in \mathcal{FP}$ such that $P \longrightarrow^* P'$

where $\longrightarrow^*$ denotes the reflexive transitive closure of $\longrightarrow$. The class of all FKS built over F is denoted by $\mathbf{FKS}(F)$.

Proposition 3. For any feature model $\mathbb{M} = (\mathbb{D}, \Phi_{ccc})$, its PPL $\mathbb{P}(\mathbb{M}) = (\mathcal{PP}(\mathbb{M}), \longrightarrow_{\mathbb{M}}, \{r\}, \mathcal{FP}(\mathbb{M}))$ is an FKS. □

Proof. According to Definition 8, this follows obviously.

Definition 10 (Feature CTL Language, fCTL). ***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 zero-arity modality ! (final), three unary CTL-operators AX, AF, AG with their usual meaning, and a unary modality $\Box$ (feature-necessity).

The *well-formed **fCTL**-formulas* ϕ are given by the grammar:

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

Other propositional connectives are defined via negation as usual in CTL: $\bot$ is $\neg\top$, etc., $\mathsf{EX}\phi$ is $\neg\, \mathsf{AX}\neg\phi$, etc., and $\Diamond\phi$ is $\neg\, \Box\neg\phi$.

The set of all **fCTL**-fromulas built over F is denoted by $\mathbf{fCTL}(F)$. □

The semantics of **fCTL**-formulas is given by the class of FKSs built over the same set of features F. Let $\mathbb{P} \in \mathbf{FKS}(F)$ be an FKS $(\mathcal{PP}, \longrightarrow, P_0, \mathcal{FP})$. We first define a satisfaction relation $\vDash$ between a product $P \in \mathcal{PP}$ and a formula $\phi \in \mathbf{fCTL}(F)$ by induction on ϕ's structure. This is done in Table 2, where we write $P \longrightarrow^! P'$ or $P' \stackrel{!}{\longleftarrow} P$ iff $P \longrightarrow^* P'$ and $P' \in \mathcal{FP}$

$P \vDash f$	iff $f \in P$ (for $f \in F$)
$P \vDash \top$	always holds
$P \vDash \neg\phi$	iff $P \vDash \phi$ does not hold
$P \vDash \phi \vee \psi$	iff $P \vDash \phi$ or $P \vDash \phi$
$P \vDash \mathsf{AX}\phi$	iff $P' \vDash \phi$ for all $P' \longleftarrow P$
$P \vDash \mathsf{AF}\phi$	iff for any path p from P, $P' \vDash \phi$ for some $P' \in p$
$P \vDash \mathsf{AG}\phi$	iff $P' \vDash \phi$ for all $P' \stackrel{*}{\longleftarrow} P$
$P \vDash !$	iff $P \in \mathcal{FP}$
$P \vDash \Box\phi$	iff $P' \vDash \phi$ for all $P' \stackrel{!}{\longleftarrow} P$

Table 2: Rules of satisfiability

Then we set $\mathbb{P} \vDash \phi$ iff $P \vDash \phi$ for all $P \in \mathcal{PP}$. We say a formula ϕ is an ***fCTL**-axiom* iff $\mathbb{P} \vDash \phi$ for all $\mathbb{P} \in \mathbf{FKS}(F)$, i.e., any FKS over F has the property specified by ϕ. Several interesting axioms and inference rules of **fCTL** are listed below.

(Persistency) $\phi \Rightarrow \mathsf{AG}\phi$
(Mand1) $\Box\phi \Rightarrow (! \Rightarrow \phi)$
(Mand2) $(! \Rightarrow \phi) \wedge (\phi \Rightarrow \Box\psi) \Rightarrow (! \Rightarrow \psi)$
(Trans1) $\Box\Box\phi \Leftrightarrow \Box\phi$
(Trans2) $(\phi \Rightarrow \Box\psi) \wedge (\psi \Rightarrow \Box\gamma) \Rightarrow (\phi \Rightarrow \Box\gamma)$

We also have distributivity of $\mathsf{AX}, \mathsf{AG}, \Box$ over $\wedge$ and $\vee$, and of AF over $\vee$.

If $\Gamma \subset \mathbf{fCTL}(F)$ is a set of formulas, we write $\Gamma \models \phi$ iff for any $\mathbb{P} \in \mathbf{FKS}(F)$, $\mathbb{P} \models \Gamma$ implies $\mathbb{P} \models \phi$. In this case we call the pair (Γ, ϕ) an ***fCTL**-inference case*. For example, **fCTL** features two schemas for generating inference cases: *modus ponens*, $\{\boldsymbol{\phi} \Rightarrow \boldsymbol{\psi}, \boldsymbol{\phi}\} \models \boldsymbol{\psi}$, and *necessity*, $\{\boldsymbol{\phi}\} \models \Box\boldsymbol{\phi}$, where bold letters denote metavariables to be substituted by formulas. Inference schemas gives rise to inference rules as usual.

The set $\mathbf{fCTL}(F)$ of formulas together with the class of all (**fCTL**-axioms and) **fCTL**-inference rules is called *(semantically defined) feature CTL*, **fCTL**. A constructive syntactical definition of **fCTL** is an open problem.

4.2 fCTL-translation of feature models

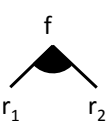
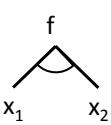
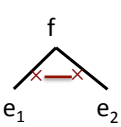
subfeature	optional feature o	mandatory feature m	OR group	XOR group	MEX group
					
$f \Leftarrow f'$ $f \Rightarrow \mathsf{EX}f'$	$f \Leftarrow o$ $f \Rightarrow \mathsf{EX}o$	$f \Leftarrow m$ $f \Rightarrow \mathsf{EX}m$ $f \Rightarrow \Box m$	$f \Leftarrow r_1 \vee r_2$ $f \Rightarrow \mathsf{EX}r_{i(i=1,2)}$ $f \Rightarrow \Box(r_1 \vee r_2)$	$f \Leftarrow x_1 \vee x_2$ $f \Rightarrow \mathsf{EX}x_{i(i=1,2)}$ $f \Rightarrow \Box(x_1 \vee x_2)$ $x_1 \wedge x_2 \Rightarrow \bot$	$f \Leftarrow e_1 \vee e_2$ $f \Rightarrow \mathsf{EX}e_{i(i=1,2)}$ $e_1 \wedge e_2 \Rightarrow \bot$

Table 3: **fCTL** translation of FD constructs

Table 3 shows how basic constructs of FDs are encoded by **fCTL**-formulas. The table is built analogously to Table 1. Formulas in the two upper rows encode subfeature relationships, the next row is for the mandatoriness requirement (if such exists) embodied into the construct, the bottom row encodes mutual exclusion if needed. In addition, the root feature adds formula $\top \Rightarrow r$. Thus any FD $\mathbb{D}$ is encoded by an **fCTL**-formula $\Phi(\mathbb{D})$, and a model $\mathbb{M} = (\mathbb{D}, \Phi_{in}, \Phi_{ex})$ is encoded by **fCTL**-formula $\Phi(\mathbb{M}) = \Phi(\mathbb{D}) \wedge \Box\Phi_{in} \wedge \Phi_{ex}$. In the next section, we will consider an example of such an encoding.

Proposition 4. *For any feature model* $\mathbb{M}$, $\mathbb{P}(\mathbb{M}) \models \Phi(\mathbb{M})$ □

Proof. Let an FM $\mathbb{M} = (\mathbb{D}, \Phi_{in}, \Phi_{ex})$. To prove this theorem, we need to show $\mathbb{P}(\mathbb{M}) = (\mathcal{FP}(\mathbb{M}), \longrightarrow_{\mathbb{M}}, \{r\}, \mathcal{FP}(\mathbb{M}))$ satisfies of each conjunctive clause in $\Phi^{ML}(\mathbb{M})$.

Consider a mandatory feature m in $\mathbb{M}$, $P\mathcal{FP}(\mathbb{M})$, and $P' \in \mathcal{FP}(\mathbb{M})$ s.t. $m^{\uparrow} \in P$ and $P \longrightarrow^{*}_{\mathbb{M}} P'$. According to persistency condition, $m^{\uparrow} \in P'$. Since $P' \models_{BS} \mathbb{M}$ and m is mandatory, $m \in P'$. This leads us to $\mathbb{P}(\mathbb{M}) \models m^{\uparrow} \Rightarrow \Box m$.

Consider an optional feature o in $\mathbb{M}$ and $P \in \mathcal{FP}(\mathbb{M})$ s.t. $o^{\uparrow} \in P$. Since $P' = P \cup \{o\}$ does not violates the conditions of Definition 6, $P' \in \mathcal{FP}(\mathbb{M})$. It is clear that, due to Definition 7, $P \longrightarrow_{\mathbb{M}} P'$. Since $\mathbb{P}(\mathbb{M})$ is an FKS (Proposition 3), there exists $P'' \in \mathcal{FP}(\mathbb{M})$ with $P' \longrightarrow^{*}_{\mathbb{M}} P''$. According to persistency condition, $o^{\uparrow}, o \in P''$. This implies $\mathbb{P}(\mathbb{M}) \models o^{\uparrow} \Rightarrow \Diamond o$.

Consider an OR-group features $G = \{r_1, \cdots, r_n\}$ in $\mathbb{M}$, $P \in \mathcal{FP}(\mathbb{M})$, and $P' \in \mathcal{FP}(\mathbb{M})$ s.t. $G^{\uparrow} \in P$ and $P \longrightarrow^{*}_{\mathbb{M}} P'$. According to Definition 3, there exists $1 \leq i \leq n$ s.t. $r_i \in P'$. This implies that $\mathbb{P}(\mathbb{M}) \models G^{\uparrow} \Rightarrow \Box(r_1 \vee \cdots \vee r_n)$.

Consider a XOR-group features $G = \{x_1, \cdots, x_n\}$ in $\mathbb{M}$, $P \in \mathcal{FP}(\mathbb{M})$, and $P' \in \mathcal{FP}(\mathbb{M})$ s.t. $G^{\uparrow} \in P$ and $P \longrightarrow^{*}_{\mathbb{M}} P'$. According to Definition 3, there exists one and only one feature from the set $\{x_1, \cdots, x_n\}$ in P'. This implies that $\mathbb{P} \models G^{\uparrow} \Rightarrow \Box(x_1 \otimes \cdots \otimes x_n)$ ($\otimes$ denotes logical xor).

Since for any $P \in \mathcal{FP}(\mathbb{M})$, $P \models_{BS} \Phi_{in}$, it follows obviously that $\mathbb{P}(\mathbb{M}) \models \Box\Phi_{in}$.

Since, according to Definition 6, any $P \in \mathcal{FP}(\mathbb{M})$ satisfies Φ_{ex}, $\mathbb{P}(\mathbb{M}) \models \Phi_{ex}$.

Since any MEX-group can be seen as optional features and exclusive constraints on the features, it is already proved. □

Note that $\mathbb{P}(\mathbb{M})$ is not the only FKS satisfying $\Phi(\mathbb{M})$. For example, consider the FD in Fig. 5. According to Table 3, its **fCTL**-theory is the conjunction of the following formulas: car, $\mathsf{car} \Rightarrow \Box\mathsf{eng}$, $\mathsf{car} \Rightarrow \Box\mathsf{trans}$, $\mathsf{trans} \Rightarrow \Box\mathsf{oil}$, $(\mathsf{eng} \Rightarrow \mathsf{car}) \wedge (\mathsf{car} \Rightarrow \mathsf{EXeng})$, $(\mathsf{trans} \Rightarrow \mathsf{car}) \wedge (\mathsf{car} \Rightarrow \mathsf{EXtrans})$, $(\mathsf{oil} \Rightarrow \mathsf{trans}) \wedge (\mathsf{trans} \Rightarrow \mathsf{EXoil})$. Its $\mathbb{P}$ is the FKS in the figure, without considering the dashed transition: the latter is excluded as it violates **I2C**. However, the FKS with the dashed transition included also satisfies the above formula. Distinguishing between PPLs and other FKSs is a non-trivial open problem.

4.3 fCTL as a CCC specification language

In this section, we show how the additional expressiveness of FML can be used for specifying PPL-properties that cannot be captured in Boolean logic.

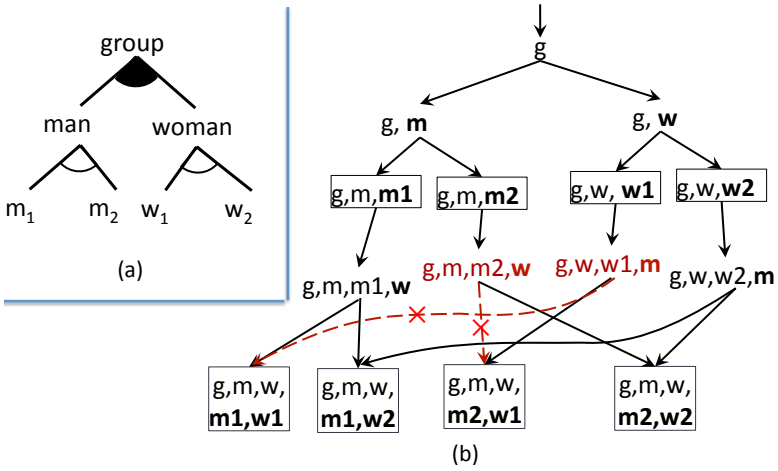


Fig. 6: Feature diagram $\mathbb{D}$ (a), and its PPL, $\mathbb{P}$ (b)

Consider the following configuration scenario. A manager wants to organize a group featuring a woman and/or a man. A man can be either m_1 or m_2. A woman can be either w_1 or w_2.

Figure 6 (a) shows the corresponding FD, $\mathbb{D}$, and Fig. 6(b) specifies its PPL $\mathbb{P}$ (ignore the two $\times$ symbols for a while), in which we abbreviate features man and woman by m and w, resp. The PPL consists of fifteen partial products, amongst which eight are full products. In particular, the four bottom final products present all possible pairs $(\mathsf{m}_i, \mathsf{w}_j)$ with $i, j = 1, 2$. The FD is encoded by **fCTL**-formula $\Phi(\mathbb{D}) = \phi_{\mathbb{D}1} \wedge \cdots \wedge \phi_{\mathbb{D}11}$ whose components are shown in Table 4. It is easy to check that $\mathbb{P} \models \Phi(\mathbb{D})$.

$\phi_{\mathbb{D}1}$	$\mathsf{group} \Leftarrow \mathsf{man} \vee \mathsf{woman}$
$\phi_{\mathbb{D}2}$	$\mathsf{group} \Rightarrow \mathsf{EXman} \wedge \mathsf{EXwoman}$
$\phi_{\mathbb{D}3}$	$\mathsf{group} \Rightarrow \Box(\mathsf{man} \vee \mathsf{woman})$
$\phi_{\mathbb{D}4}$	$\mathsf{man} \Leftarrow \mathsf{m}_1 \vee \mathsf{m}_2$
$\phi_{\mathbb{D}5}$	$\mathsf{man} \Rightarrow \mathsf{EXm}_1 \wedge \mathsf{EXm}_2$
$\phi_{\mathbb{D}6}$	$\mathsf{man} \Rightarrow \Box(\mathsf{m}_1 \vee \mathsf{m}_2)$
$\phi_{\mathbb{D}7}$	$\mathsf{m}_1 \wedge \mathsf{m}_2 \Rightarrow \bot$
$\phi_{\mathbb{D}8}$	$\mathsf{woman} \Leftarrow \mathsf{w}_1 \vee \mathsf{w}_2$
$\phi_{\mathbb{D}9}$	$\mathsf{woman} \Rightarrow \mathsf{EXw}_1 \wedge \mathsf{EXw}_2$
$\phi_{\mathbb{D}10}$	$\mathsf{woman} \Rightarrow \Box(\mathsf{w}_1 \vee \mathsf{w}_2)$
$\phi_{\mathbb{D}11}$	$\mathsf{w}_1 \wedge \mathsf{w}_2 \Rightarrow \bot$
ϕ_{C1}	$\mathsf{w}_1 \wedge \mathsf{man} \Rightarrow \neg\mathsf{EXm}_1$
ϕ_{C2}	$\mathsf{m}_2 \wedge \mathsf{woman} \Rightarrow \neg\mathsf{EXw}_1$

Table 4: **fCTL** encoding of FD in Fig. 6(a)

Now suppose that the manager wants to respect preferences of group members for the selection of their coworkers. Suppose that man m_1 and woman w_2 do not have any concerns, but man m_2 wants to work with woman w_2, and woman w_1 prefers to work with man m_2. To address these concerns, the following two (cross-cutting) constraints are to be added to $\mathbb{D}$:

(**C1**) If woman w_1 is selected, and a man is needed, then it must be m_2.

(**C2**) If man m_2 is selected, and a woman is needed, then it must be w_2.

These constraints are encoded by the two lower **fCTL**-formulas in Table 4. FKS $\mathbb{P}$ does not satisfy them: the product, at which the curved dashed $\times$-labeled transitions begins, violates the formula ϕ_{C1}, and the product at which the straight dashed $\times$-transition begins violates ϕ_{C2} (these products are red with a color display). However, the actual violators are transitions rather than products,

and in order to adapt $\mathbb{P}$ to satisfy the constraints, it is sufficient to remove both $\times$-transitions. Let $\mathbb{P}^{\times}$ denotes the resulting FKS, then $\mathbb{P}^{\times} \models \Phi(\mathbb{D}) \wedge \phi_{C1} \wedge \phi_{C2}$.

Since PPLs $\mathbb{P}^{\times}$ and $\mathbb{P}$ have the same partial and full products, their difference cannot be captured by the Boolean logic. However, **fCTL**-constraints can affect the set of full products, if the way the latter are designed is taken into account. For example, the manager can decide first to select a woman for the group, and then proceed with the man selection. Then the constraint (C1) would make the full product including $\{\mathsf{m}_1, \mathsf{w}_1\}$ unreachable. Thus, **fCTL** allows us to express useful configuration constraints not expressible in the Boolean logic.

5 Related and Future Work

Kripke structures and concurrency modeling. There is an enormous body of work about Kripke Structures (KS) and their associated logics. The underlying frames are typically cyclic graphs, and there are possible different states with the same set of true propositions. Our FKS are much simpler: the frames are DAGs, labeling is injective and actually redundant, and the Singletonity condition makes transitions very simple. Similar structures under the name *Event KS* were studied in [11]. However, the transition relation in event KS coincides with set inclusion.

More general than EKS are Configuration Structures (CS) [19], in which transitions are set inclusions but not the converse (like in our FKS). This makes event structures (a syntax for CS) somewhat similar to FD (a syntax for FKS), but with instantiation constraints formulated in a different language. Also, the **I2C**-requirement was not specifically considered, at least, explicitly. Investigation of relations between CS and FKS in more detail is part of our future work.

An important feature of FKS distinguishing them amongst general KS is the presence of a special subset of final states. On the other hand, the notion of accepting states is well known in automata and formal language theory. And indeed, in a forthcoming paper we will elaborate an automata-based view of FKS.

Use of FM in behavior modeling. In a well-known paper [4], Classen et al. study model checking of a family of transition systems. Such a family is modeled by what they call *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.

Staged configuration. Czarnecki et al. [6] introduce the concept of *staged configuration* in which the process of specifying a product is performed in stages, such that in each stage some configuration choices are eliminated. In other words, in each step a specialization of the given FD is generated. This process is continued until a fully specialized feature diagram denoting only one configuration

is obtained. Our setting is somewhat dual: rather than eliminating a feature in syntactical processing of FDs, we augment partial products with features until a full product (a final configuration) is reached. We plan to investigate this duality in future work.

Algebraic modeling of feature modeling. Höfner et al. developed a semiring model for product families [12]. An FD is encoded as a term in the semiring signature, and semantics is provided by the semiring of product families (whose carrier is the set of all PLs over a feature set). 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. But one important distinction can be stated immediately: for Höfner et al., a product is a set of prime features—leaves in the feature tree, while non-leaf features are derived terms; in contrast, we follow a common FM-practice and consider all features in the tree to be basic. Also, 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 [1,8]. Their setting is much more concrete than ours: features are blocks of code, or other components, whose composition makes a product. This work focus on operation of feature/delta composition and delta management.

6 Conclusion

We have presented a behavioral 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 product lines encompassing partial products satisfying the constraints are Kripke structures of a special kind that we called FKS. We have also demonstrated that properties of FKSs, and hence partial product lines, can be described by a suitable version of modal logic, namely, **fCTL**. These results establish close connections between feature modeling and behavior modeling, which we believe can be fruitful for both fields.

References

1. Sven Apel, Christian Lengauer, Bernhard Möller, and Christian Kästner. An algebraic foundation for automatic feature-based program synthesis. *Sci. Comput. Program.*, 75(11):1022–1047, 2010.
2. Don Batory. *Feature models, grammars, and propositional formulas.* Springer, 2005.
3. David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 2010.
4. Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay, and Jean-François 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*, pages 335–344. ACM, 2010.
5. Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.
6. Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration using feature models. In *Software Product Lines*, pages 266–283. Springer, 2004.
7. Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.
8. Ferruccio Damiani, Luca Padovani, and Ina Schaefer. A formal foundation for dynamic delta-oriented software product lines. In Klaus Ostermann and Walter Binder, editors, *GPCE*, pages 1–10. ACM, 2012.
9. Magnus Eriksson, Jürgen Börstler, and Kjell Borg. The pluss approach: domain modeling with features, use cases and use case realizations. In *SPLC 2005*, SPLC'05, pages 33–44, Berlin, Heidelberg, 2005. Springer-Verlag.
10. Rohit Gheyi, Tiago Massoni, and Paulo Borba. Automatically checking feature model refactorings. *J. UCS*, 17(5):684–711, 2011.
11. Vineet Gupta and Vaughan R. Pratt. Gages accept concurrent behavior. In *FOCS*, pages 62–71. IEEE Computer Society, 1993.
12. Peter Höfner, Ridha Khédri, and Bernhard Möller. An algebra of product families. *Software and System Modeling*, 10(2):161–182, 2011.
13. Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document, 1990.
14. Kyo C Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. Form a feature oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5(1):143–168, 1998.
15. Mike Mannion. Using first-order logic for product line model validation. In *Software Product Lines*, pages 176–187. Springer, 2002.
16. Klaus Pohl, Günter Böckle, and Frank Van Der Linden. *Software product line engineering: foundations, principles, and techniques*. Springer, 2005.
17. Matthias Riebisch, Kai Böllert, Detlef Streitferdt, and Ilka Philippow. Extending feature diagrams with uml multiplicities. In *IDPT 2002, Pasadena, CA*, volume 50. Citeseer, 2002.
18. Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wasowski, and Krzysztof Czarnecki. Reverse engineering feature models. In *ICSE 2011*, pages 461–470. IEEE, 2011.
19. Rob J. van Glabbeek and Gordon D. Plotkin. Configuration structures. In *LICS*, pages 199–209, 1995.