GSDLAB TECHNICAL REPORT

Modeling Product Lines with Kripke Structures and Modal Logic (Extended Version)

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

GSDLAB–TR 2015-04-01 April 2015

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
McMaster University
University of Waterloo
zdiskin@gsd.uwaterloo.ca

Aliakbar Safilian
McMaster University
safiliaa@mcmaster.ca

Tom Maibaum
McMaster University
maibaum@mcmaster.ca

Shoham Ben-David
University of Waterloo
shohambd@gmail.com

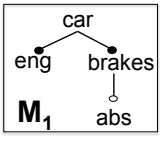
***Abstract*—Product lines are now an established framework for software design. They are specified by special diagrams called *feature models*. For formal analysis, the latter are usually encoded by propositional theories with Boolean semantics. We discuss a major deficiency of this semantics, and show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

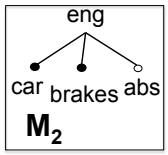
I. Introduction

The *Software Product Line* approach is well-known in the software industry. Products in a product line (PL) share some common *mandatory* features, and differ by having some *optional* features that allow the user (or developer) to configure the product the user wants (e.g., MS Office, a Photoshop, or the Linux kernel). Instead of producing a multitude of separate products, the vendor designs a single PL encompassing a variety of products, which results in a significant reduction in development time and cost [16].

Industrial PLs may be based on thousands of features interrelated in complex ways [14]. Methods of specifying PLs and checking the validity of a PL against a specification is an active research area. Elite software engineering conferences like ICSE, ASE, and FM, readily accept papers on PL [2], [17], [20]; there are conference series specially devoted to PL, such as Generative Programming and Component Engineering (GPCE) and the Software Product Line Conference (SPLC), and several textbooks about PL have been written [7], [16].

The most common method for designing a PL is to build a *feature model (fm)* of the products.[1] A toy example is shown in the inset figure. Model M_1 says that a (root feature called) car *must* have an engine and brakes (black bullets denote *mandatory* subfeatures), and brakes can *optionally* (note the hollow bullet) be equipped with an anti-skidding system. The model specifies a PL consisting of two products: $P = \{\mathsf{car}, \mathsf{eng}, \mathsf{brakes}\}$ and $P' = P \cup \{\mathsf{abs}\}$. fms of industrial size can be big and complex, require tools for their management and analysis, and thus should be represented by formal objects processable by tools. A common approach is to consider features as atomic propositions, and view an fm as a theory in the Boolean propositional logic (BL), whose valid valuations are to be exactly the valid products defined by the fm [3]. For example, model M_1 represents BL theory $\Phi(M_1) = \{\mathsf{eng}\to\mathsf{car}, \mathsf{brakes}\to\mathsf{car}, \mathsf{abs}\to\mathsf{brakes}\} \cup \{\mathsf{car}\to\mathsf{eng}, \mathsf{car}\to\mathsf{brakes}\}$: the first three implications encode subfeature dependencies (a feature can appear in a product only if its parent is in the product), and the last two implications encode the mandatory dependency between features. This approach gave rise to a series of prominent applications for analysis of industrial size PLs [9], [11], [18]. However, the Boolean semantics for fms has an almost evident drawback of misrepresenting fms' hierarchial structure. Indeed, the second inset figure shows an fm M_2 that is essentially different from M_1 (and is, in fact, pathological), but has the same set of products, $PL(M_2) = PL(M_1) = \{P, P'\}$ determined by the same Boolean theory $\Phi(M_2) = \{\mathsf{car}\to\mathsf{eng}, \mathsf{brakes}\to\mathsf{eng}, \mathsf{abs}\to\mathsf{eng}\} \cup \{\mathsf{eng}\to\mathsf{car}, \mathsf{eng}\to\mathsf{brakes}\} = \Phi(M_1)$: only grouping of implications has changed, but it is immaterial for BL. The core of the problem is that two different dependencies are encoded by the same construct (implication), and hence are not semantically distinguished.





We are not the first to have noticed this drawback, e.g., it is explicitly mentioned in [18] (where fms' semantics not captured by BL is called *ontological*), and many other researchers and practitioners in the field are probably aware of the situation. Nevertheless, as far as we know, no alternative to the Boolean logic of feature modeling (FM) has been proposed in the literature, which we think is theoretically and conceptually unsatisfactory. Even more importantly, inadequate logical foundations for FM hinder practical analyses: as important information contained in fms is not captured by their BL-encoding, this information is either missing from analyses, or treated informally, or hacked in an ad hoc way. In a sense, this is yet another instance of the known software engineering problem, when semantics is hidden in the application code

[1] We use small letters for this acronym (fms will abbreviate the plural form) as below we will use FM to abbreviate *feature modeling*.

rather than explicated in the specification, with all its negative consequences for software testing, debugging, maintenance, and communication between the stakeholders.

The main goal of the paper is to show that Kripke structures and modal logic provide an adequate logical basis for FM. Our main observation is that the key notion of FM—a product built from features—should be considered an *instantiation process* rather than its final result. We call intermediate states of this process *partial products*, and argue that what an fm M really specifies is a partially ordered set of products that we call a *partial product line*, *PPL(M)*; the commonly considered products of M (we call them *full* or *final*) only form a subset of *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 a *feature KS (fKS)*. The latter are specifiable by a suitable version of modal logic, which we call *feature CTL* (denoted by fCTL), as it is basically a fragment of CTL enriched with a zero-ary modality that only holds in states representing full products. We show that any fm M can be represented by an fCTL theory $\Phi_{\mathsf{ML}}(M)$ accurately specifying M's intended semantics: the main result of the paper states that for any fKS K, $K \models \Phi_{\mathsf{ML}}(M)$ iff $K = PPL(M)$, and hence $\Phi_{\mathsf{ML}}(M)$ is a sound and complete representation of the fm. Then we can replace fms by the respective fCTL-theories, which are highly amenable to formal analysis and automated processing.

In a broader perspective, we want to show that mathematical foundations of FM are mathematically interesting, and to attract the attention of the Theoretical Computer Science community to the area. We will describe several problems that we believe are mathematically interesting and practically useful. Especially intriguing are connections of FM to concurrency modeling. In fact, PLs can be seen as a special interpretation of configuration structures [21]: features are events, partial products are configurations, and PPLs are configuration structures; fms can then be seen as a far reaching generalization of Winskel's event structures and other formalisms for specifying dependencies between events. It appears that the syntactical aspects of specifying concurrency (including transaction mechanisms), i.e., having a convenient and suggestive notation suitable for practitioners, have not received much attention in concurrency modeling. This is where we believe FM can make a non-trivial contribution. We will discuss some details in Sect. VII. On the other hand, we would like to have the paper readable by an FM researcher, and to convince her that the logic of FM 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 only then explore it formally.

Our plan for the paper is as follows. The next section motivates the formal framework developed in the paper: we describe the basics of FM, and show how the deficiency of the Boolean semantics can be fixed by introducing *partial* products and transitions between them. In Sect. III, the notions of fms and PPLs they generate are formalized. In Sect. IV, we introduce the notion of fKSs as immediate abstraction of PPLs, and fCTL as a language to specify fKS properties. We show, step-by-step, how to translate an fm into an fCTL-theory, and prove our main result in Sect. V. In Sect. VI, we discuss some practical applications of the modal logic view of fms. We discuss the connection between fms and event modelling of concurrent systems in Sect. VII. Other related work is discussed in Sect. VIII. Conclusion of the paper and several interesting open problems are discussed in Sect. IX.

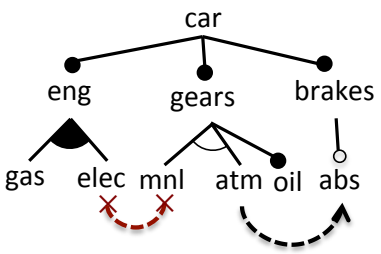


Figure 1: An fm

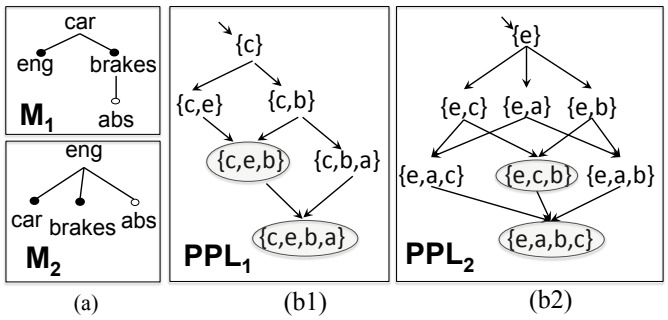
II. Feature Models and Partial Product Lines

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

A. Basics of Feature Modeling

A *feature model* is a graphical structure presenting a hierarchial decomposition of features with some possible *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. Black angles denote so called *OR-groups*: an engine can be either gas (gasoline), or ele (electric), or both. Hollow angles denote *XOR-groups* (eXclusive OR): a gear is either mnl (manual) or atm (automatic) but not both; it must be supplied with oil as dictated by the black-bullet edge. The $\times$-ended arc says that an electric engine cannot be combined with a manual gear, and the arrow-headed arc says that an automatic gear requires ABS. According to the model, the set of features {car, eng,gas, gear,mnl,oil, brakes} is a valid product, but replacing the gasoline engine by electric, or removal of oil, would make the product invalid. In this way, the model compactly specifies seven valid products amongst the set of 2^9 possible combinations of 9 non-root features (the root is always included), and exhibits dependencies between choices.

In the Boolean view of feature modeling, an fm is a representation of a propositional Boolean theory. For example, the theory encoded by the model in Fig. 1 consists of



(a) (b1) (b2)

Figure 2: From fms to PPLs: simple cases

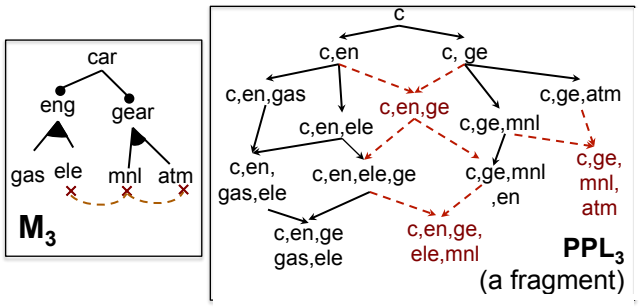


Figure 3: From fms to PPLs: Complex case

a set of implications denoting subfeature dependencies and unary mandatory dependencies, as explained in the introduction, plus three implications denoting grouped mandatoriness: $\{\mathsf{eng}\to\mathsf{gas}\vee\mathsf{ele},\ \mathsf{gear}\to\mathsf{mnl}\vee\mathsf{atm},\ \mathsf{mnl}\wedge\mathsf{atm}\to\bot\}$ (with $\bot$ denoting False), plus two implications encoding CCCs: $\{\mathsf{gas}\wedge\mathsf{ele}\to\bot,\mathsf{atm}\to\mathsf{abs}\}$. However, as we have seen above, a BL encoding is deficient, and in the next subsection we will present a different view of FM semantics.

B. PPL semantics: Products as Processes.

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 an arrow and write, e.g., $\{\mathsf{c}\}\longrightarrow_{\mathsf{e}}\{\mathsf{c},\mathsf{e}\}$, 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 fm, $PPL(M_2)$, 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 both structurally (6 nodes and 7 edges in PPL_1 versus 8 nodes and 12 edges in PPL_2), and in the content of products (e.g., products {c} and {c,b} present in PPL_1 but absent in PPL_2, whereas {e} and {e,a} are present in PPL_2 but absent from PPL_1) too. This essential difference between PPLs properly reflects the essential difference between the fms.

C. PPL semantics: From lattices to transition systems

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

The forgotten mandatoriness of some features appears as incompleteness of some objects; we call them *proper* partial products. Partial products closed under mandatoriness are *full* (sometimes we will say *final*). Thus, PPLs of simple fms such 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.

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 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 (ignore the difference between solid and dashed edges for a while); to ease reading, the number of letters in the acronym for a feature corresponds to its level in the tree, e.g., c stands for car, en for eng etc.

Now let us consider how the additional structure embodied in the 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.

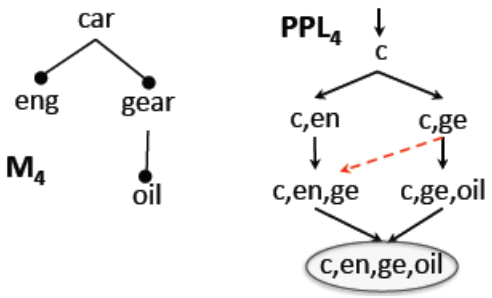


Figure 4

To specify this lattice-theoretically, we add to the lattice of features a universal *bottom* element $\perp$ (a feature to be a subfeature of any feature), and write two defining equations: $\mathsf{ele} \wedge \mathsf{mnl} = \perp$ and $\mathsf{mnl} \wedge \mathsf{atm} = \perp$. Then, in the filter lattice, the formal down-join of products {c,en,ele,ge} and {c,ge,mnl,en} "blow up" and become equal to the set of all features ("False implies everything"). The same happens with the other pair of conflicting products.

Next we consider the mandatoriness structure of model M_3 (given by black bullets and triangles). This structure determines a subset of the PPL consisting of full products (not shown in Fig. 3) as we discussed above. In addition, mandatoriness affects the set of valid partial products as well. Consider the product $P = \{\mathsf{c}, \mathsf{en}, \mathsf{ge}\}$ at the center of the diagram. The left instantiation path leading to this product, $\{\mathsf{c}\} \longrightarrow_{\mathsf{en}} \{\mathsf{c}, \mathsf{en}\} \longrightarrow_{\mathsf{ge}} P$ is not good because gear was added to engine *before* the latter is fully assembled (a mandatory choice between being electric or gasoline, or both, has still not been made). Jumping to another branch from inside of the branch being processed is poor design practice that should be prohibited, and the corresponding transition is declared invalid. Similarly, transition $\{\mathsf{c}, \mathsf{ge}\} \longrightarrow_{\mathsf{en}} P$ 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 with model M_4 and its PPL. In the latter, the "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), and 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.

Also, since modal logic is more expressive than Boolean logic, it provides a more expressive language for CCCs over fms. Later in Sect. VI, we will provide an example in which some CCCs cannot be expressed by BL, but our modal logic.

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

In the next two sections, we make the constructs discussed above formal.

III. FMs and PPLs formally

Our plan for this section is as follows. In Sect. III-A, we give a formal definition of a feature model, whose structure is carefully chosen to support our work in the paper. Sect. III-B defines a propositional logic encoding of an fm, and the corresponding notions of a full and a partial products. Sect. III-C formally defines an fm's PPL as a transition systems.

A. Feature Trees and Feature Models

Typical fms are trees with some 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 either OR-groups (e.g., {gas,ele}) 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* is a pair $T_{\mathcal{OR}} = (T, \mathcal{OR})$ of the following components.

(i) $T = (F, r, \_^{\uparrow})$ is a tree whose nodes are *features*: F denotes the set of all features, $r \in F$ is the root, and function $\_^{\uparrow}$ maps each non-root feature $f \in F_{\_r} \stackrel{\text{def}}{=} F \setminus \{r\}$ to its parent $f^{\uparrow}$. The inverse function that assigns to each feature the set of its children (called *subfeatures*) is denoted by $f_{\downarrow}$; this set is empty for leaves. It is easy to see that the set of f's siblings is the set $(f^{\uparrow})_{\downarrow} \setminus \{f\}$. The set of all ancestors and all descendants of a feature f are denoted by $f^{\uparrow\uparrow}$ and $f_{\downarrow\downarrow}$, 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 $\{G \subset F\colon\ G \neq \emptyset$ and $f \# g$ for all $f, g \in G\} \subset 2^F$.

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

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

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

Exclusive and inclusive 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. Note that an fm is a purely syntactic object contrary to the common usage of term 'model' in logic.

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

B. Full and Partial Products

A common approach to formalizing the PL (of full products) of a given fm is to use Boolean propositional logic [3]. Features are considered as atomic propositions, and dependencies between features are specified by logical formulas. For example, if a feature f' is a subfeature of feature f, we have an implication $f' \rightarrow f$ (if a product has feature f', it must have feature f as well). If $\{g_1, g_2\}$ is an OR-group of f's subfeatures, we write $f \rightarrow (g_1 \vee g_2)$; if, in addition, features g_1, g_2 are mutually exclusive, we write $g_1 \wedge g_2 \rightarrow \bot$. In this way, given an fm $M = (T, \mathcal{OR}, \mathcal{EX}, \mathcal{IN})$, each of its four components gives rise to a respective propositional theory as shown in the upper four rows of Table I: later we will discuss the four theories in detail and explain the !-superscripts; the subscript BL is needed because later we will also consider modal theories encoded by fms.[2] Together these theories constitute theory $\Phi^{!}_{\mathsf{BL}}(M)$, and a set of features P is a legal full product for M iff $P \models \Phi^{!}_{\mathsf{BL}}(M)$ Since publishing the seminal paper [14], this propositional view on feature modeling became common and has been used in both theoretical and practice-oriented work [3], [9], [18].

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

(1)	$\Phi_{\mathsf{BL}}(T) = \{\top \rightarrow r\} \cup \{f' \rightarrow f\colon\ f \in F, f' \in f_{\downarrow}\}$
(2)	$\Phi_{\mathsf{BL}}(\mathcal{EX}) = \{\bigwedge G \rightarrow \bot\colon\ G \in \mathcal{EX}\}$
(3!)	$\Phi^{!}_{\mathsf{BL}}(\mathcal{OR}) = \{f \rightarrow \bigvee G\colon\ f \in F, G \in \mathcal{OR}(f)\}$
(4!)	$\Phi^{!}_{\mathsf{BL}}(\mathcal{IN}) = \{\bigwedge G \rightarrow \bigvee G'\colon\ (G, G') \in \mathcal{IN}\}$
(all!)	$\Phi^{!}_{\mathsf{BL}}(M) = \Phi_{\mathsf{BL}}(T) \cup \Phi_{\mathsf{BL}}(\mathcal{EX}) \cup \Phi^{!}_{\mathsf{BL}}(\mathcal{OR}) \cup \Phi^{!}_{\mathsf{BL}}(\mathcal{IN})$
(3)	$\Phi^{\mathsf{I2C}}(T_{\mathcal{OR}}) = \{f \wedge g \rightarrow (\bigwedge \Phi^{!}(T^{f}_{\mathcal{OR}})) \vee (\bigwedge \Phi^{!}_{\mathsf{BL}}(T^{g}_{\mathcal{OR}})) : f, g \in F, f^{\uparrow} = g^{\uparrow}\}$
(all)	$\Phi_{\mathsf{BL}}(M) = \Phi_{\mathsf{BL}}(T) \cup \Phi_{\mathsf{BL}}(\mathcal{EX}) \cup \Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$

Below we revise the propositional encoding of fms based on our discussion in Sect. II: we introduce two propositional theories for, resp., partial and full products (subsection B.1) and show how the I2C-principle can be propositionally encoded (subsection B.2).

B.1) Enabling vs. Causality.

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

The subfeature relationship is fundamental, and any product having a subfeature f' but missing its superfeature f should be considered ill-formed; we can say that superfeature f *enables* its subfeature f' and all reasonable products must respect enabling. In contrast, if f' is a mandatory subfeature of f, a product having f but missing f' is just incomplete rather than ill-formed. We can say that feature f *causes* f' so that partial products violating causality are possible, and only full product must respect it. [3]

Thus, we have two Boolean theories for the same fm M. One is its *instantiation theory* $\Phi_{\mathsf{BL}}(M)$ that encodes the basic structural dependencies a well-formed product must satisfy, and thus defines all partial products. This theory consists of three componets as specified in row (all) in the Table: $\Phi_{\mathsf{BL}}(T)$ is the BL-encoding of subfeature dependen-

[2] $\bigvee G$ and $\bigwedge G$ are conjunction and disjunction of all formulas in a set of formulas G.

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

cies (row (1), $\Phi_{\mathsf{BL}}(\mathcal{EX})$ is the BL-encoding of exclusive dependencies (row (2)), and in section B.2 we will consider yet another ingredient—the Boolean encoding of the I2C-condition, $\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$. The other propositional theory, M's *full product theory* $\Phi^{!}_{\mathsf{BL}}(M)$ consists of four components: $\Phi_{\mathsf{BL}}(T)$ and $\Phi_{\mathsf{BL}}(\mathcal{EX})$ as above, plus BL-encoding $\Phi^{!}_{\mathsf{BL}}(\mathcal{OR})$ of the mandatoriness dependencies embodied in the $\mathcal{OR}$-structure (row ($3^{!}$), plus BL-encoding $\Phi^{!}_{\mathsf{BL}}(\mathcal{IN})$ of the inclusive CCCs (row($4^{!}$)), which we treat as mandatory for only full products rather than affecting instantiation (i.e., as causal rather than enabling). With a more refined approach to feature modeling, a CCC should be labeled as either causal or enabling, but with the current FM practice, CCCs are not labeled and we thus consider them as causal, i.e., constraining full products only.

Definition *3:* **(Full Products).** A *full product* over an fm $M = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ is a set of features $P \subseteq F$ satisfying theory $\Phi^{!}_{\mathsf{BL}}(M)$ defined in Table I.

The set of all full products is called M's *full product set* and denoted by $\mathcal{FP}_M$. Thus, $\mathcal{FP}_M = \{P \subseteq F\colon\ P \models \Phi^{!}_{\mathsf{BL}}(M)\}$. □

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

B.2) Instantiate to Completion via Propositional Logic.

Consider once again PPL_3 in Fig. 3, from which product $\{\mathsf{c}, \mathsf{en}, \mathsf{ge}\}$ is excluded as violating the I2C principle. Note that in order to specify this exclusion propositionally, we *cannot* declare that features en and ge are mutually exclusive and write $\{\mathsf{en}\wedge\mathsf{ge} \rightarrow \bot\}$ because down the lattice they are combined in product $\{\mathsf{c}, \mathsf{en}, \mathsf{ele}, \mathsf{ge}\}$ below $\{\mathsf{c}, \mathsf{en}\}$, and in product $\{\mathsf{c}, \mathsf{ge}, \mathsf{mnl}, \mathsf{en}\}$ below $\{\mathsf{c}, \mathsf{ge}\}$ as well. In other words, the conflict between features en and ge is transient rather than permanent, and its propositional specification is not trivial. We solve this problem by introducing the notion of a *feature subtree* induced by a feature in Definition 4, and then specifying theory $\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$ shown in row (3) in Table I. The theory 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 *4:* **(Induced Subfeature Tree and** I2C**).** Let $T_{\mathcal{OR}} = (T, \mathcal{OR})$ be a feature tree over a set of features F, and $f{\in}F$. A *feature subtree induced* by f is a pair $T^{f}_{\mathcal{OR}} = (T^{f}, \mathcal{OR}^{f})$ with T^{f} being the tree under f, i.e., $T^{f} \stackrel{\text{def}}{=} (f_{\downarrow\downarrow} \cup \{f\}, f, \_^{\uparrow})$, and mapping $\mathcal{OR}^{f}$ is inherited from $\mathcal{OR}$, i.e., for any $g \in f_{\downarrow\downarrow}$, $\mathcal{OR}^{f}(g) = \mathcal{OR}(g)$. □

Definition *5:* **(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 instantiation theory $\Phi_{\mathsf{BL}}(M)$ specified in row (all) in Table I. (Recall that a full product is a set of features satisfying theory $\Phi^{!}_{\mathsf{BL}}(M)$.) We denote the set of all partial products by $\mathcal{PP}_M$. Thus, $\mathcal{PP}_M = \{P \subseteq F\colon\ P \models \Phi_{\mathsf{BL}}(M)\}$.

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

Proposition *1:* For any feature model M, $\Phi^{!}(M) \models \Phi(M)$. Hence, full products as defined in Definition 3 form a subset of partial products, $\mathcal{FP}(M) \subseteq \mathcal{PP}(M)$.

Proof: Note that $\Phi^{!}(\mathcal{OR}) \models \{f \rightarrow \bigwedge \Phi^{!}(T^{f}_{\mathcal{OR}}) : f{\in}F\}$ and hence $\Phi^{!}(\mathcal{OR}) \models \Phi^{\mathsf{I2C}}(\mathcal{OR})$. ■

Note that transition exclusion discussed in Sect. II-C cannot be explained with Boolean logic and needs a modal logic; we will give a suitable logic and show how it works in Sect. V.

C. 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* or *step) transition*.

Two necessary conditions are obvious: the parent $f^{\uparrow}$ must be in P, and f should not be in conflict with features in P, that is, $P' \models \big(\Phi_{\mathsf{BL}}(T) \cup \Phi_{\mathsf{BL}}(\mathcal{EX})\big)$. Compatibility with I2C is more complicated.

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

$$(3)_{P,f} \qquad \Phi^{\mathsf{I2C}}_{\mathsf{BL}}(P, f) \stackrel{\text{def}}{=} \bigcup \{\Phi^{!}_{\mathsf{BL}}(T^{g}_{\mathcal{OR}})\colon\ g \in P \cap (f^{\uparrow})_{\downarrow}\}$$

where $T^{g}_{\mathcal{OR}}$ denotes the subtree induced by feature g as described in Definition 4. (Note that set $P \cap (f^{\uparrow})_{\downarrow}$ may be empty, and then theory $\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(P, f)$ is also empty.)

We say P is *fully instantiated wrt.* f if $P \models \Phi^{\mathsf{I2C}}_{\mathsf{BL}}(P, f)$. □

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

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

(a) $P' = P \uplus \{f\}$ for some feature $f{\notin}P$ such that the following three conditions hold: (a1) $P' \models \Phi_{\mathsf{BL}}(T)$, (a2) $P' \models \Phi_{\mathsf{BL}}(\mathcal{EX})$, and (a3) $P \models \Phi^{\mathsf{I2C}}_{\mathsf{BL}}(P, f)$.

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

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

The following result is important.

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

Proof: If $P' = P$, the proposition is obvious. Consider now the case of $P' = P \uplus \{f\}$ with $P' \models \Phi_{\mathsf{BL}}(T) \cup \Phi_{\mathsf{BL}}(\mathcal{EX})$ and $P \models \Phi^{\mathsf{I2C}}_{\mathsf{BL}}(P, f)$. We need to prove that $P' \models \Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$.

Let $g{\in}P$ be an arbitrary feature with $g^{\uparrow}{=}f^{\uparrow}$, i.e., $g \in P \cap (f^{\uparrow})_{\downarrow}$. By definition of relative fullness, if $P \models \Phi^{\mathsf{I2C}}_{\mathsf{BL}}(P, f)$, then definitely $P \models \Phi^{!}_{\mathsf{BL}}(T^{g}_{\mathcal{OR}})$ (one of the union's components). This implies $P' \models \Phi^{!}_{\mathsf{BL}}(T^{g}_{\mathcal{OR}})$, and hence $P' \models \bigcup \{\Phi^{!}_{\mathsf{BL}}(T^{g}_{\mathcal{OR}})\colon\ g{\in}P, g^{\uparrow}{=}f^{\uparrow}\}$. The above statement, along with $P \models \Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$, imply that $P' \models \{f{\wedge}g \rightarrow (\bigwedge \Phi^{!}_{\mathsf{BL}}(T^{f}_{\mathcal{OR}})) \vee (\bigwedge \Phi^{!}_{\mathsf{BL}}(T^{g}_{\mathcal{OR}}))\colon f, g \in F, f^{\uparrow}{=}g^{\uparrow}\}$. ■

Definition *8:* **(Partial Product Line).** Let $M = (T_{\mathcal{OR}}, \mathcal{EX}, \mathcal{IN})$ be an fm. The *partial product line* determined by M is a triple $\mathbb{P}(M) = (\mathcal{PP}_M, \longrightarrow_M, I_M)$ with the set $\mathcal{PP}_M$ of partial products given by Definition 5, transition

relations $\longrightarrow_M$ given by Definition 7 (so that full products, and only them, are equipped with self-loops), and the initial product $I_M = \{r\}$ consisting of the root feature. □

IV. Feature Kripke Structures and Their Logic

In this section, we introduce *Feature Kripke Structures* (fKS), which are an immediate abstraction of PPLs generated by fms. Then we introduce a modal logic called *feature CTL* (fCTL), which is tailored for specifying fKSs's properties.

A. Feature Kripke Structures

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

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

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

The components of an fKS K are subscripted with $_K$ if needed, e.g., $\mathcal{PP}_K$. We denote the class of all fKSs built over a set of features F by $\mathbf{K}(F)$. Note that any product in an fKS eventually evolves into a full product because F is finite, $\longrightarrow$ is left-total, and all loops are self-loops. Hence, an fKS enjoys the following:
(*Finality*) For all $P \in \mathcal{PP}$, there exists a full product P' such that $P \longrightarrow^+ P'$.

We will also need the notion of a sub-fKS of an fKS.

Definition *10:* **(Sub-fKS).** Let K, K' be two fKSs. We say K is a sub-fKS of K', denoted by $K \sqsubseteq K'$, iff $I_K = I_{K'}$, $\mathcal{PP}_K \subseteq \mathcal{PP}_{K'}$, and $\longrightarrow_K \subseteq \longrightarrow_{K'}$. □

The following proposition is an obvious corollary of Definition 8.

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

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 $\mathbf{K}(F)$ is too big. We want to characterize $\mathbb{P}(M)$ in a more precise way by defining as small as possible a class of fKSs to which $\mathbb{P}(M)$ would provably belong. Hence, we need a logic for defining classes of fKSs by specifying a fKS's properties.

Table II: 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 $P \longrightarrow P$

B. Feature Computation Tree Logic

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

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

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

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

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

We define $K \models \phi$ iff $P \models \phi$ for all $P \in \mathcal{PP}_K$ (equivalently, iff $I_K \models \mathsf{AG}\phi$).

For instance, consider Group Example (Fig. 5 in Sect. **??**), with self-loops at full (framed) products. The following statements ϕ hold in K, i.e., $K \models \phi$, for $\phi = \mathsf{g}$, $\phi =! \rightarrow \mathsf{w} \vee \mathsf{m}$, $\phi = \{\mathsf{g}, \mathsf{w}\} \rightarrow \mathsf{AX}(\mathsf{w}_1 \vee \mathsf{w}_2)$, $\phi = (\mathsf{w}_1 \wedge \mathsf{m} \wedge \neg\mathsf{m}_1) \rightarrow \mathsf{AX}\neg\mathsf{m}_1$, $\phi = (\mathsf{m}_2 \wedge \mathsf{w} \wedge \neg\mathsf{w}_1) \rightarrow \mathsf{AX}\neg\mathsf{w}_1$. The last two formulas encode CCCs **C1** and **C2** resp., described in the Example.

V. fCTL-theory of Feature Models

In this section we prove our main results. Given an fm M over a finite set of features F, we build two fCTL theories from M's data, $\Phi_{\mathsf{ML}\subseteq}(M)$ and $\Phi_{\mathsf{ML}}(M)$ (index ML refers to Modal Logic), such that the former theory is a subset of the latter, and the following holds for any fKS $K\in\mathbf{K}(F)$:

Theorem *1 (Soundness):* $\mathbb{P}(M) \models \Phi_{\mathsf{ML}}(M)$.

Theorem *2 (Semi-completeness):*
$K \models \Phi_{\mathsf{ML}\subseteq}(M)$ implies $K \sqsubseteq \mathbb{P}(M)$.

Theorem *3 (Completeness):* $K \models \Phi_{\mathsf{ML}}(M)$ iff $K = \mathbb{P}(M)$.

We have discussed the value of Completeness in the introduction. Semi-completeness is useful (as an auxiliary intermediate step to completeness, but also) for some important practical problems in FM such as *specialization* [19] (M is a specialization of another fm M' if $\mathcal{FP}_M \subset \mathcal{FP}_{M'}$), and some other analysis operations [4] over fms. These operations are normally considered for full product lines (FPLs) only, but can be redefined for PPLs as well.

We will build theories $\Phi_{\mathsf{ML}\subseteq}(M)$ and $\Phi_{\mathsf{ML}}(M)$ from small *component* theories, which specify the respective properties of M's PPL in terms of fCTL. Before we proceed to defining these theories and proofs, in order to provide some guidance through the proofs, in Section A we discuss the structure of the entire component family, and explain how the compound theories, $\Phi_{\mathsf{ML}\subseteq}(M)$, $\Phi_{\mathsf{ML}}(M)$, and $\Phi_{\mathsf{ML}+}(M) \stackrel{\text{def}}{=} \Phi_{\mathsf{ML}}(M) \setminus \Phi_{\mathsf{ML}\subseteq}(M)$ are built from them. Then, in Section B, we zoom into component theories and explain how they are built. Section C presents the proofs.

A. Structure of the component family

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

Rows of the table are indexed by structural *concerns* to be logically encoded; columns are named by the goals of these encodings: to provide semi-completeness wrt. FPL and PPL (split into Boolean and modal components), and to provide completeness wrt. FPL and PPL: a theory in the last column is the union of all theories in its row, and thus ensures completeness wrt. the concern corresponding to the row. Each internal theory is an encoding of the corresponding concern for the corresponding goal. For example, theory $\Phi^{!}_{\mathsf{ML}\subseteq}(\mathcal{OR})$ modally specifies the $\mathcal{OR}$ structure to provide semi-completeness wrt. FPL (note the ! superindex). For another example, $\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$ is a Boolean encoding of the I2C-principle, and its neighbor on the right is the additional modal constraint for the same concern—it is needed to ensure semi-completeness. The empty neighbor on the right means that nothing should be added (for this concern) to ensure completeness. We do not intend to make the table strictly logical: its goal is to reference component theories and explain their intentions.

B. The content of component theories

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

The theory $\Phi^{\downarrow}_{\mathsf{ML}+}(T)$ states that if a feature f is visited in a current state (partial product) without visiting any of its children, say g, then there must be another state immediately accessible from the current state visiting g. The union of this theory and $\Phi_{\mathsf{BL}}(T)$ generates a completeness theory $\Phi_{\mathsf{ML}}(T)$. An fKS K satisfying $\Phi_{\mathsf{ML}}(T)$ is guaranteed to capture the tree structure T.

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

The theories corresponding to $\mathcal{OR}$ and $\mathcal{IN}$ have the same nature: both deal with full products (states with self-loop transitions). The theory $\Phi^{!}_{\mathsf{ML}\subseteq}(\mathcal{OR})$ is the modal version of the Boolean theory $\Phi^{!}_{\mathsf{BL}}(\mathcal{OR})$ (Table I). Consider an OR group G. The theory $\Phi^{!}_{\mathsf{ML}\subseteq}(\mathcal{OR})$ states that if the G's parent is visited in a current state, then at least one of the elements involved in G must be visited in any final products accessible from the current state. The theory $\Phi^{!}_{\mathsf{ML}\subseteq}(\mathcal{IN})$ is the modal version of Boolean theory $\Phi^{!}_{\mathsf{BL}}(\mathcal{IN})$. Let (G, G') be an inclusive constraint. The theory $\Phi^{!}_{\mathsf{ML}\subseteq}(\mathcal{IN})$ states that if all the elements involved in G are visited in a current state, then at least one of the elements in G' must be visited in any final products accessible from the current state. Obviously, these two theories are derivable from the theory $\Phi^{!}_{\mathsf{ML}\subseteq}(M)$.

The theories $\Phi^{!}_{\mathsf{ML}\subseteq}(M)$ and $\Phi^{!}_{\mathsf{ML}+}(M)$ are clear (see Table IV). $\Phi^{!}_{\mathsf{ML}\subseteq}(M)$ holding in an fKS guarantees that any full product in the fKS is a full product of M. On the other hand, any fKS satisfying the theory $\Phi^{!}_{\mathsf{ML}}(M)$ ($= \Phi^{!}_{\mathsf{ML}\subseteq}(M) \cup \Phi^{!}_{\mathsf{ML}+}(M)$) must include all full products of M and only them.

Recall that the theory $\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$ (Table I) guarantees that the partial products of the PPL of M respect the I2C principle. However, as discussed in Sect. II-C, transitions also have to respect this principle. The modal theory $\Phi^{\mathsf{I2C}\not\to}_{\mathsf{ML}\subseteq}(T_{\mathcal{OR}})$ excludes the invalid transitions due to the I2C principle (see Table IV). This theory states that if a feature is visited in a current state without being completely instantiated, then there must not be any states immediately accessible from the current state including one of the feature's siblings. Then, the completeness theory relating to I2C, $\Phi^{\mathsf{I2C}}_{\mathsf{ML}}(T_{\mathcal{OR}})$, would be the union of

Table IV: Definitions of (basic) fCTL theories

$\Phi^{\downarrow}_{\mathsf{ML}+}(T) = \{f \wedge \neg \bigvee f_{\downarrow} \to \mathsf{EX}g : f, g \in F, g^{\uparrow} = f\}$

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

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

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

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

$\Phi^{\mathsf{I2C}\not\to}_{\mathsf{ML}\subseteq}(T_{\mathcal{OR}}) = \{f \wedge \neg\bigwedge\Phi^{!}(T^{f}_{\mathcal{OR}}) \to \neg\mathsf{EX}g : f, g \in F, f^{\uparrow} = g^{\uparrow}\}$

$\Phi^{\leftrightarrow}_{\mathsf{ML}+}(T_{\mathcal{OR}}, \mathcal{EX}) = \{\bigwedge \Phi^{\mathsf{I2C}}(f) \wedge \neg f \wedge \neg \bigvee \Phi^{\mathcal{EX}}(f) \to \mathsf{EX}f : f \in F\}$, where

$\Phi^{\mathsf{I2C}}(f) = \{g \to \Phi^{!}(T^{g}_{\mathcal{OR}}) : g, f \in F, g^{\uparrow} = f^{\uparrow}, g \neq f\}$

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

Table III: Component and Compound Theories

M	Semi-completeness		To Ensure Completeness	Completeness
	BL	ML		
T	$\Phi_{\mathsf{BL}}(T)$	$\varnothing$	$\Phi^{\downarrow}_{\mathsf{ML}+}(T)$	$\Phi_{\mathsf{ML}}(T)$
$\mathcal{EX}$	$\Phi_{\mathsf{BL}}(\mathcal{EX})$	$\varnothing$	$\varnothing$	$\Phi_{\mathsf{ML}}(\mathcal{EX})$
$\mathcal{OR}$	$\varnothing$	$\Phi^{!}_{\mathsf{ML}\subseteq}(\mathcal{OR})$	$\varnothing$	$\Phi^{!}_{\mathsf{ML}}(\mathcal{OR})$
$\mathcal{IN}$	$\varnothing$	$\Phi^{!}_{\mathsf{ML}\subseteq}(\mathcal{IN})$	$\varnothing$	$\Phi^{!}_{\mathsf{ML}}(\mathcal{IN})$
I2C	$\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$	$\Phi^{\mathsf{I2C}\nrightarrow}_{\mathsf{ML}\subseteq}(T_{\mathcal{OR}})$	$\varnothing$	$\Phi^{\mathsf{I2C}}_{\mathsf{ML}}(T_{\mathcal{OR}})$
$\mathcal{FP}_M$	$\varnothing$	$\Phi^{!}_{\mathsf{ML}\subseteq}(M)$	$\Phi^{!}_{\mathsf{ML}+}(M)$	$\Phi^{!}_{\mathsf{ML}}(M)$
$\mathcal{PP}_M$	$\Phi_{\mathsf{BL}}(M)$	$\varnothing$	$\Phi^{\downarrow}_{\mathsf{ML}+}(T)\cup\Phi^{\leftrightarrow}_{\mathsf{ML}+}(T_{\mathcal{OR}},\mathcal{EX})$	$\Phi^{\circ}_{\mathsf{ML}}(M)$
$\mathbb{P}(M)$	$\Phi_{\mathsf{ML}\subseteq}(M)$		$\Phi_{\mathsf{ML}+}(M)$	$\Phi_{\mathsf{ML}}(M)$

$\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(T_{\mathcal{OR}})$ and $\Phi^{\mathsf{I2C}\nrightarrow}_{\mathsf{ML}\subseteq}(T_{\mathcal{OR}})$.

Recall that, according to Definition 5, a set of features is a valid partial product iff it satisfies the Boolean theory $\Phi_{\mathsf{BL}}(M)$. However, any fKS satisfying this theory does not necessarily include all valid partial products. To ensure that the fKS includes all partial products, we add modal theories $\Phi^{\downarrow}_{\mathsf{ML}+}(T)$ and $\Phi^{\leftrightarrow}_{\mathsf{ML}+}(T_{\mathcal{OR}},\mathcal{EX})$. Consider a state P and a feature f such that $f\notin P$. The theory $\Phi^{\leftrightarrow}_{\mathsf{ML}+}(T_{\mathcal{OR}},\mathcal{EX})$ states that if adding f to P does not violate the exclusive constraints and I2C principle, then there must be an immediately accessible state from P including f. The corresponding completeness theory is denoted by $\Phi^{\circ}_{\mathsf{ML}}(M)$ and is equal to $\Phi_{\mathsf{BL}}(M)\cup\Phi^{\downarrow}_{\mathsf{ML}+}(T)\cup\Phi^{\leftrightarrow}_{\mathsf{ML}+}(T_{\mathcal{OR}},\mathcal{EX})$.

Any fKS satisfying the semi-completeness theory $\Phi_{\mathsf{ML}\subseteq}(M)$ would be a substructure of $\mathbb{P}(M)$. On the other hand, the theory $\Phi_{\mathsf{ML}}(M)$, which is the union of $\Phi_{\mathsf{ML}\subseteq}(M)$ and $\Phi_{\mathsf{ML}+}(M)$, guarantees completeness, i.e., any fKS K satisfying $\Phi_{\mathsf{ML}}(M)$ is equal to the PPL of M.

C. Proofs

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

Soundness: $\mathbb{P}(M)\models\Phi_{\mathsf{ML}}(M)$.

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

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

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

Let $P\in\mathcal{PP}_M$ and $P\models f\wedge\neg\bigvee f_{\downarrow}$ and $g\in f_{\downarrow}$. We want to show that $P\models\mathsf{EX}g$. Let $P'=P\cup\{g\}$. According to (a), $P\models\Phi_{\mathsf{BL}}(T)\cup\Phi_{\mathsf{BL}}(\mathcal{EX})$. Since the g's parent is already in P', adding g to P does not violate $\Phi_{\mathsf{BL}}(T)$. Since exclusive constraints are defined on incomparable features, adding g to P also does not violate $\Phi_{\mathsf{BL}}(\mathcal{EX})$. Therefore, $P'\models\Phi_{\mathsf{BL}}(T)\cup\Phi_{\mathsf{BL}}(\mathcal{EX})$. Since all subfeatures of f are absent in P, $\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(P,f)=\varnothing$ (note Definition 6) and hence $P\models\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(P,f)$. Since $P'\models\Phi_{\mathsf{BL}}(T)\cup\Phi_{\mathsf{BL}}(\mathcal{EX})$ and $P\models\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(P,f)$, according to Definition 7, there is a transition $P\longrightarrow_M P'$. Therefore, $P\models\mathsf{EX}g$.

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

(d) $\mathbb{P}(M)\models\Phi^{\mathsf{I2C}\nrightarrow}_{\mathsf{ML}\subseteq}(T_{\mathcal{OR}})$ follows obviously. Indeed, this theory guarantees that there would not be an invalid transition due to I2C principle.

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

Let f and P be a feature and a partial product of M, respectively, such that $f\notin P$, $P\models\Phi^{\mathsf{I2C}}(f)$, and $P\not\models\bigvee\Phi^{\mathcal{EX}}(f)$. Thus, according to Definition 7, there exists a transition $P\longrightarrow_M P\cup\{f\}$, which implies $P\models\mathsf{EX}\ f$. This results in $\mathbb{P}(M)\models\Phi^{\leftrightarrow}_{\mathsf{ML}+}(T_{\mathcal{OR}},\mathcal{EX})$.

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

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

Proof: Let $K\models\Phi_{\mathsf{ML}\subseteq}(M)$. $I_K=I_M$, since, due to $K\models\Phi_{\mathsf{BL}}(T)$, $K\models r$ (r is the root feature of M).

Since $K\models\Phi_{\mathsf{BL}}(M)$, according to Definition 5, $\mathcal{PP}_K\subseteq\mathcal{PP}_M$.

Now, we are going to show that $\longrightarrow_K\subseteq\longrightarrow_M$.

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

Consider a transition $P\longrightarrow_K P'$, where $P'=P\cup\{f\}$ for a feature $f\notin P$. We want to show that there is a transition $P\longrightarrow_M P'$ in $\mathbb{P}(M)$. Again, note that any state in K is a partial product of M. To prove this statement, according to Definition 7, we need to show that (a1) $P'\models\Phi_{\mathsf{BL}}(T)$, (a2) $P'\models\Phi_{\mathsf{BL}}(\mathcal{EX})$, and (a3) $P\models\Phi^{\mathsf{I2C}}_{\mathsf{BL}}(P,f)$. (a1) and (a2) is an immediate corollary of $K\models\Phi_{\mathsf{BL}}(M)$. To prove (a3), we need to show that for any siblings g with $g\in P$, $P\models\Phi^{!}_{\mathsf{BL}}(T^{g}_{\mathcal{OR}})$ (see Definition 6). Assume by a way of contradiction that

$P \not\models \Phi^{!}_{\mathsf{BL}}(T^{g}_{\mathcal{OR}})$, i.e., g is not completely instantiated in P. Since $K \models \Phi^{\mathsf{I2C}\rightarrow}_{\mathsf{ML}\subseteq}(T_{\mathcal{OR}})$, $g \in P$, and $P \not\models \Phi^{!}_{\mathsf{BL}}(T^{g}_{\mathcal{OR}})$, there must not be a transition $P \longrightarrow_K P'$. This leads us to a contradiction. Thus, (a3) holds.

Based on the above reasonings, $\longrightarrow_K \subseteq \longrightarrow_M$. ■

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

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

Proof: Let $K \models \Phi^{\circ}_{\mathsf{ML}}(M)$. By Theorem 2, $\mathcal{PP}_K \subseteq \mathcal{PP}_M$. 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 instantiated 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 traversal would result in the sequence $\langle \mathsf{car}, \mathsf{gear}, \mathsf{mnl}, \mathsf{oil}, \mathsf{eng} \rangle$.

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

The following condition (R) holds:

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

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

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

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

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

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

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

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

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

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

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

According to Theorem 2, $\longrightarrow_K \subseteq \longrightarrow_M$. Now what we need is to prove that any non-loop transition in $\mathbb{P}(M)$ is also a transition in K. Note that, according to Lemma 1, $\mathcal{PP}_K = \mathcal{PP}_M$. Consider a transition $P \longrightarrow_M P'$, where $P' = P \cup \{f\}$ for a feature $f \notin P$. We want to show that there is a transition $P \longrightarrow_K P'$ in K. According to Definition 7, $P' \models \Phi_{\mathsf{BL}}(T) \cup \Phi_{\mathsf{BL}}(\mathcal{EX})$, and $P \models \Phi^{\mathsf{I2C}}_{\mathsf{BL}}(P, f)$. Thus, there are two choices:

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

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

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

(ii): Since $P' \models \Phi_{\mathsf{BL}}(\mathcal{EX})$, $P \models \neg \bigvee \Phi^{\mathcal{EX}}(f)$. Also, $P \models \Phi^{\mathsf{I2C}}_{\mathsf{BL}}(P, f)$ implies that $P \models \Phi^{!}_{\mathsf{BL}}(T^{g}_{\mathcal{OR}})$ for any $g \in P \cap (f^{\uparrow})_{\downarrow}$, which means $P \models \Phi^{\mathsf{I2C}}(f)$. Hence, due to $\Phi^{\leftrightarrow}_{\mathsf{ML}+}(T_{\mathcal{OR}}, \mathcal{EX})$, 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. Hence, $\longrightarrow_M \subseteq \longrightarrow_K$. ■

VI. Practical applications

In this section, we discuss some concrete tasks in FM, which would be improved by the use of modal logic view of fms. These tasks are grouped into (a) *fm analysis*, (b) *PL-builder* vs. *PL-client view*, and (c) *reverse engineering* of fms.

A. Automated Analysis of Feature Models

Due to some practical applications, several analysis problems over fms arise in practice. Some operations deal with only one given fm. For example, for a given fm, we may want to determine: the *core features*, i.e., those features that are in all full products of the fm; the *least common ancestor* (LCA) of a given set of features; all *subfeatures* of a given feature. Some other operations are given two fms and investigate their relationships. For example, given two fms M and M', we may want to decide whether: they are *refactoring* or not, i.e., they are semantically equivalent; M is a *specialization* of M' or not, i.e., M subsume M' in the semantics sense.

We could group analysis operations into two types: *syntactic* and *semantic*. Syntactic analysis operations rely on fms' structures, i.e., they are ask some questions about the hierarchical structure of a given fm. For example, the LCA and subfeatures operations (see above) are in this type. On the other hand, semantic analysis operations, as the name suggests, rely on fms' semantics. For example, the refactoring and specialization operations are in this kind.

Clearly, semantic analysis based on the Boolean semantics can be erroneous, e.g., the two fms in the introduction are

semantically equivalent (refactoring) in the Boolean view, but not in the modal view!.

Normally, practitioners use both types of analysis and thus deal with pairs $(M, Sem(M))$. $Sem(M)$ denotes the semantics of M. In the Boolean view, they must keep both components, which complicates maintenance: you change M, but forget to change $Sem(M)$, or the other way round. Moreover, industrial fms may have thousands of features and many dependencies between them [18]. As industrial fms may be huge, practitioners sometimes use off-the-shelf tools (e.g., SAT-solvers) to perform even syntactic analysis on $Sem(M)$ rather than M, and thus are prone to errors. In the modal view, these problems disappear as the Modal semantics of M, i.e., $\mathbb{P}(M)$, keeps all information we need.

B. PL-builder vs. PL-client View

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

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 5(a) shows the corresponding fm, M, and Fig. 5(b) represents its PPL, in which we abbreviate features man and woman by m and w (ignore the difference between dashed and other transitions for a while). The PPL consists of fifteen partial products, amongst which eight are full products (they are framed). In particular, the four bottom final products present all possible pairs $(\mathsf{m}_i, \mathsf{w}_j)$ with $i, j = 1, 2$.

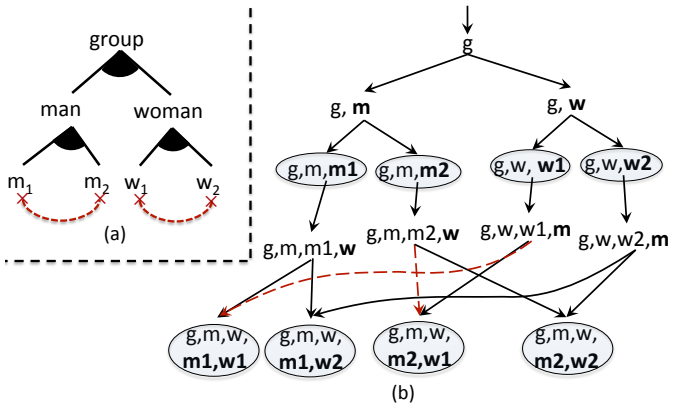


Figure 5: Feature model M (a), and its PPL (b)

Now suppose that the manager wants to respect preferences of the first selected member for the selection of her/his coworkers. Suppose that the man m_1 and the woman w_2 do not have any concerns, but the man m_2 wants to work with the woman w_2, and the woman w_1 prefers to work with the man m_2. To address these concerns, the following two CCCs are to be added: (**C1**) If the woman w_1 is selected, and a man is needed, then it must be m_2; (**C2**) If the man m_2 is selected, and a woman is needed, then it must be w_2. Obviously, these constraints rely on the order of making feature selection.

Let us see how the above constraints influence the PPL. **C1** excludes the transition $\{\mathsf{g}, \mathsf{w}, \mathsf{w}_1, \mathsf{m}\} \longrightarrow \{\mathsf{g}, \mathsf{m}, \mathsf{w}, \mathsf{m}_1, \mathsf{w}_1\}$, since w_1 in the source does not allow the manager to pick m_1 as her colleague. In a similar way, **C2** excludes the transition $\{\mathsf{g}, \mathsf{m}, \mathsf{m}_2, \mathsf{w}\} \longrightarrow \{\mathsf{g}, \mathsf{m}, \mathsf{w}, \mathsf{m}_2, \mathsf{w}_1\}$. The excluded transitions are represented by dashed arrows in Fig. 5. Note that the above CCCs do not make any partial product from the PPL illegal. They only exclude some transitions. This is why they cannot be expressed using Boolean logic: encoding these constraints by Boolean implications gives us $\mathsf{w}_1 \to \mathsf{m}_2$ and $\mathsf{m}_2 \to \mathsf{w}_2$, which implies $\mathsf{w}_1 \to \mathsf{w}_2$ and contradicts the exclusion constraint $\mathsf{w}_1 \wedge \mathsf{w}_2 \to \bot$.

Consider the Group Example, with self-loops at full (framed) products. The following statements ϕ hold in K, i.e., $K \models \phi$, for $\phi = \mathsf{g}$, $\phi =! \to \mathsf{w} \vee \mathsf{m}$, $\phi = \{\mathsf{g}, \mathsf{w}\} \to \mathsf{AX}(\mathsf{w}_1 \vee \mathsf{w}_2)$, $\phi = (\mathsf{w}_1 \wedge \mathsf{m} \wedge \neg \mathsf{m}_1) \to \mathsf{AX}\neg \mathsf{m}_1$, $\phi = (\mathsf{m}_2 \wedge \mathsf{w} \wedge \neg \mathsf{w}_1) \to \mathsf{AX}\neg \mathsf{w}_1$. The last two formulas encode CCCs **C1** and **C2** resp., described in the Example.

C. Reverse Engineering of fms

Reverse engineering of fms is an active research area in FM. The problem statement is as follow: given a PL, we want to build an appropriate fm representing the PL. Depending on the representation of the given PL, the current approaches are grouped into two kinds: reverse engineering of fms from BL formulas [9], reverse engineering of fms from textual descriptions of features [**?**], [**?**]. She et al. in [18] argue that non of these approaches are complete. Indeed, the main challenge of this task is to determine an appropriate hierarchical structure of features. The BL approach is incomplete, since, as already discussed, the BL semantics cannot capture the hierarchical structure of the features. The textual approach is also not desirable for two reasons: it is an informal approach, and also "it suggests only a single hierarchy that is unlikely the desired one" [18]. To relieve the deficiencies of these approaches, the current stat-of-the-art approach [18] proposes a heuristic based approach in which both types of inputs are given as input. However, if we take the given input to be the fCTL theory of the PL (in other words, its PPL), reverse engineering of fms becomes simpler and better manageable. This is because the given fCTL theory contains everything needed to build the corresponding fm. Also, our careful decomposition of fms' structure and theories into small blocks is because it would allow better tuning of the reverse engineering process.

VII. Feature Modeling vs. Event-based Concurrency Modeling

In this section, we summarize similarities and differences between FM and event-based concurrency modeling. (We will use the following abbreviations: EM for Event Modeling, and em for an event model.) We will also point to several possibilities of fruitful interactions between the two disciplines.

Following the survey in [21], we distinguish three approaches in EM. The first is based on a topological notion of

Table V: Event vs. feature modeling

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

a configuration structure (CS) $(E,\mathcal{C})$ with E a set (possibly infinite) of *events*, and $\mathcal{C}\subset 2^E$ a family of subsets (usually finite) of events, which satisfy some closure conditions (e.g., under intersection and directed union). Sets from $\mathcal{C}$ are called *configurations* and understood as states of the system: $X\in\mathcal{C}$ denotes a state in which all events from X have already occurred. In the second approach, valid configurations are specified indirectly by some structure $\boldsymbol{D}$ of dependencies between events, which make some configurations invalid. Formally, some notion of *validity* of a set $X\subset E$ wrt. $\boldsymbol{D}$ is specified so that an *event structure* (ES) $(E,\boldsymbol{D})$ determines a CS $\{X\subset E\colon\ X$ is *valid* wrt. $\boldsymbol{D}\}$. Typical representatives of this approach are Winskel's prime and general event structures. The third approach is an ordinary encoding of sets of propositions by Boolean logical formulas. Then an em is just a Boolean theory, i.e., a pair (E,Φ) with Φ a set of propositional formulas over set E of propositions. The left half of Table V summarizes the rough mini-survey above.

Notably, for a typical CS, transitions between states are a derived notion; e.g., in [12], any set inclusion is a transition; in [21], special conditions are to hold in order for a set inclusion to be a valid transition. In contrast, *event automata (EA)* by Pinna and Poigné [15] are tuples $(E,\mathcal{C},\rightarrow,I)$ with $\rightarrow$ a *given* transition relation over configurations (states), and $I\in\mathcal{C}$ an initial state.

FM is quite directly related to EM, and actually can be seen as a special interpretation of EM. Indeed, features can be considered as events, (partial) products as configurations, and fms as special ESs (see the middle row in the table, where $M=(T_{\mathcal{OR}},\mathcal{EX},\mathcal{IN})$). An important distinction of the Boolean FM is the presence of a special subset of *final* states (products), so that FM's topological and logical counterparts are triples rather than pairs (see the Boolean column in the table, where we slightly changed notation to save space and fit the table into one text column: the subindex BL is replaced by reference to the model M). Pinna and Poigné [15] mention final states (they call them *quiescent*) but do not actually use them, whereas for FM, final products are a crucial ingredient. The last column of the table describes FM's basic topological and logical structures in the modal logic view: the upper row is our fKS, and the bottom one is the theory specified in Sect. V. Our fKS is exactly an EA with quiescent states (= full products or self-looped states), which, additionally, satisfies two conditions: left-totality of transitions and Self-loop Only (Singletonicity is satisfied in EAs), but Pinna and Poigné do not apply modal logic for specifying EA's properties (and do not even mention it), and do not consider the I2C-principle.

The comparison above shows enough similarities and differences to hope for a fruitful interaction between the two fields. We are currently investigating what FM can usefully bring to EM; and can mention several simple findings. The presence of two separate Boolean theories allows us to formally distinguish between enabling and causality (Sect. III-B); we are also not aware of a propositional specification of transient conflicts such as our Boolean and modal encoding of I2C. Moreover, such encoding is nothing but a compact formal specification of a transaction mechanism, which is usually considered to be non-trivial. Overall, we have shown that a simple formalism of fms is capable of encoding very complex modal theories with a wealth of information. Using this formalism and notation for EM is, perhaps, the most exciting possible application of FM to concurrency modeling.

VIII. Related and Future Work

A. Staged Configuration

Czarnecki et al. [8] introduced the concept of *staged configuration* in which a product instantiation of a given fm M is performed in a sequence of stages. In each stage a *specialization* of M is generated by making some choices.

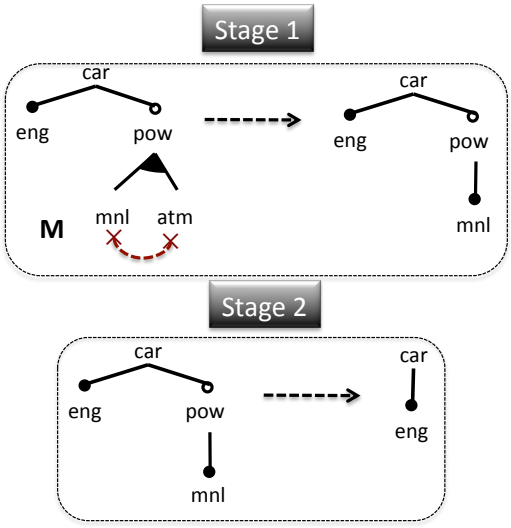


Figure 6: Staged Configuration

This process, called the *product derivation process* (PDP), is continued until a fully specialized fm denoting only one configuration is obtained. Fig. 6 shows an fm M (pow stands for power lock) and a PDP for the full product {car, eng} of this model. It includes two stages: in the first stage, the choice between manual and automated power lock is made, and in the second stage, the power lock is eliminated. The corresponding sequence of stages is called a configuration path [6].

Fig. 7 shows the PPL of M. We call a path starting at the initial and ending at a full product an *instantiation path*. Although both instantiation and configuration paths show how to instantiate full products, they are essentially

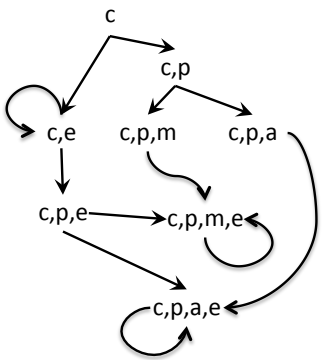


Figure 7: PPL

different. Firstly, configuration paths are sequences of fms, while instantiation paths are sequences of products. Secondly, in staged configuration, one can make choices irrespective of any conditions other than exclusive constraints, while an instantiation path shows how to reach a full product by including the features step by step in a top-down fashion not violating the I2C-principle. For example, consider also Fig. 6: in the first stage, we make choices between mnl and atm before making the decision whether the car is equipped with a power locker or not. Such a decision is not allowed in PPLs: a feature cannot occur into a product without its parent.

B. 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 [13]. 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 a 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 PL of a given fm$M$ is encoded as a term in the PL algebra generated over the *basic features* of M; the latter are leaves in the feature diagram. This is an important (meta)feature in the approach, which is in contrast to a common FM practice. As an example, consider the following feature diagram, which is adopted from [13]. The encoded term corresponding to this fm is as follows: $car = (manual + automatic) \times horsepower \times (1 + aircondition)$.

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.

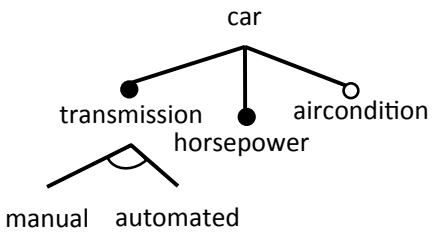


Figure 8: adopted from [13]

C. 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 study.

IX. Conclusion and Open Problems

Conclusion. We have presented a novel view on fms, in which a product is an instantiation process rather than its final result. We called such products *partial*, and showed that the set of partial products together with a set of (carefully defined) valid transitions between them can be considered as a special Kripke structure, whose properties are specifiable by a special fragment of CTL enriched with a constant modality. We called the logic fCTL. Our main result shows that an fm can be considered as a compact representation of a rather complex fCTL-theory. Thus, the logic of FM is modal rather than Boolean. We have discussed several concrete tasks in feature modeling, which would be improved by the use of modal logic view of fms. These tasks include analysis of fms, reverse engineering of fms, and the developer vs. client view.

Open Problems. There exist several interesting open problems in the modal logic view of fms, which would be mathematically and practically important. We briefly discuss some of these problems in the following:

(*i*) *Complete Axiomatic System for* fCTL. Finding a complete axiomatic system for fCTL is theoretically interesting. It would be also important in practice (see (b) below).

(*ii*) *Automated Analysis of fms*. To implement analysis operations over a given fm M, one could apply either a *model checker* or *theorem prover*. To apply a model checker, we would need to transform M to its PPL $\mathbb{P}(M)$ and express given analysis problems into fCTL. We plan to implement the analysis operations over some realistic examples using some existing model checking tools. To take advantage of theorem provers, we first need to have a complete axiomatic system for our logic. There exist some theorem provers such as CTL-RP [22] and MLSolver [10], which can be used for reasoning about the CTL formulae.

(*iii*) *Characterization of the class of fKSs produced by the class of fms*. One of the questions that have been left open in the paper, is to axiomatically definition of the class of fKSs produced by the class of fms. We plan to address this problem.

(*iv*) *Process Algebras for fKSs and fms*. Industrial systems are usually very complex and the companies design their systems by utilizing smaller systems, which themselves are produced by other companies [1]. Therefore, their corresponding fms could be seen as a compound of several smaller fms. In this sense, proper process algebras for defining complex fms and their corresponding fKSs become fundamental and essential.

(*v*) *Strong version of* I2C. Recall that the current version of the I2C principle says that two incompariable features can be included together in a partial product if at least one of them has been already completely instantiated. The current version of this principle is unavoidable, if we would like to realize the step-by-step computation. Note that this is why fKSs are enforced to satisfy the singletonicity condition (see Definition 9). However, in some contexts, it also makes sense to consider a *stronger* version of I2C principle: *two incomparable features can be included together in a partial product if both of them have been already completely instantiated*. Indeed, we plan to specify such a stronger version of the I2C-principle, in which a full product instantiation is always a transaction (which corresponds to replacing disjunction by conjunction in the definition of theory $\Phi_{\mathsf{BL}}^{\mathsf{I2C}}(T_{\mathcal{OR}})$, row (3) in Table I).

(*vi*) *A New Modal Logic view of Event-based modeling*. We have discussed the intriguing similarities between EM and FM in Sect. VII. We consider investigating a new ML view of ever-based models as one of our future tasks.

REFERENCES

[1] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. Managing multiple software product lines using merging techniques. *France: UniversityofNiceSophiaAntipolis. TechnicalReport, ISRN I3S/RR*, 6, 2010.

[2] Sven Apel, Christian Kästner, and Christian Lengauer. Featurehouse: Language-independent, automated software composition. In *ICSE*, pages 221–231, 2009.

[3] Don Batory. *Feature models, grammars, and propositional formulas*. Springer, 2005.

[4] 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.

[5] 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.

[6] Andreas Classen, Arnaud Hubaux, and Patrick Heymans. A formal semantics for multi-level staged configuration. *VaMoS*, 9:51–60, 2009.

[7] Paul C. Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, August 2001.

[8] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration using feature models. In *Software Product Lines*, pages 266–283. Springer, 2004.

[9] Krzysztof Czarnecki and Andrzej Wasowski. Feature diagrams and logics: There and back again. In *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 23–34. IEEE, 2007.

[10] Oliver Friedmann and Martin Lange. A solver for modal fixpoint logics. *Electronic Notes in Theoretical Computer Science*, 262:99–111, 2010.

[11] Rohit Gheyi, Tiago Massoni, and Paulo Borba. Automatically checking feature model refactorings. *J. UCS*, 17(5):684–711, 2011.

[12] Vineet Gupta. Concurrent kripke structures. In *Proceedings of the North American Process Algebra Workshop*, 1993.

[13] Peter Höfner, Ridha Khédri, and Bernhard Möller. An algebra of product families. *Software and System Modeling*, 10(2):161–182, 2011.

[14] Mike Mannion. Using first-order logic for product line model validation. In *Software Product Lines*, pages 176–187. Springer, 2002.

[15] G. Michele Pinna and Axel Poigné. On the nature of events: Another perspective in concurrency. *Theor. Comput. Sci.*, 138(2):425–454, 1995.

[16] Klaus Pohl, Günter Böckle, and Frank Van Der Linden. *Software product line engineering: foundations, principles, and techniques*. Springer, 2005.

[17] Fabricia Roos-Frantz, David Benavides, and Antonio Ruiz-Cortés. Feature model to orthogonal variability model transformation towards interoperability between tools. *KISS@. ASE, New Zealand*, 2009.

[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] Thomas Thum, Don Batory, and Christian Kastner. Reasoning about edits to feature models. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 254–264. IEEE, 2009.

[20] Salvador Trujillo, Don S. Batory, and Oscar Díaz. Feature oriented model driven development: A case study for portlets. In *ICSE*, pages 44–53, 2007.

[21] Rob J. van Glabbeek and Gordon D. Plotkin. Configuration structures. In *LICS*, pages 199–209, 1995.

[22] Lan Zhang, Ullrich Hustadt, and Clare Dixon. A refined resolution calculus for ctl. In *Automated Deduction–CADE-22*, pages 245–260. Springer, 2009.